LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

Datu piekļuves objekti SQL datubāzēm

BAKALAURA DARBS

Autors: **Andris Balodis**
Stud. apl. Nr. ab08498
Darba vadītājs: Dr.dat. Jānis Zuters

RĪGA 2013

ANOTĀCIJA

Darbā tiek sīkāk izpētīti datu piekļuves objekti SQL datubāzēm. Tiek apzināti iemesli, kāpēc datu piekļuves objekti netiek tik plaši izmantoti programmu izstrādē kā tiešās piekļuves metode un katras metodes priekšrocības un trūkumi. Darba gaitā tiek izveidoti vienkārši Java piemēri, lai labāk izprastu gan programmēšanas sarežģītības pakāpi, gan patērēto laiku izstrādājot programmas, kas izmanto datu piekļuves objektus un SQL datubāzes.

Atslēgvārdi: *SQL, Datu piekļuves objekti*

ABSTRACT

This research is a more detailed examination of data access objects for SQL databases. It identifies the reasons why the data access objects are not as widely used in the development as direct access method and each method advantages and disadvantages. Int course of research there are created simple Java examples to better understand the complexity of both programming and the time spent developing programs that use the data access objects and SQL databases.

Keywords: *SQL, Data access objects*

SATURS

IEVADS

Mūsdienās vairums izstrādāto programmu izmanto kādu datu glabāšanas metodi, piemēram, datubāzes. Izstrādājot programmatūru, datubāzu vaicājumi bieži vien tiek rakstīti tieši programmatūras pirmkodā. Taču pastāv arī iespēja visus datubāzu vaicājumus ievietot atsevišķās klasēs, kuras tiek dēvētas par datu piekļuves objektiem. Tomēr biežāk programmatūras izstrādē tiek izmantota tiešās piekļuves metode. Tāpēc darbā tiek noskaidroti iemesli, kāpēc tiek izmantota vai nu tiešā datu piekļuve, vai nu datu piekļuves objekti.

Bakalaura darbs ir sadalīts trīs nodaļās. Pirmajā nodaļā notiek iepazīšanās ar datu piekļuves objektiem un to teorētiskais apraksts. Otrajā nodaļā notiek iepazīšanās ar tiešās piekļuves metodi un tās teorētiskais apraksts. Trešajā nodaļā ir aprakstīts kā tiek veikta programmu izstrāde un to analīze. Ceturtajā nodaļā tiek veikta kopējās salīdzināšanas analizēšana un veikti secinājumi par metodēm.

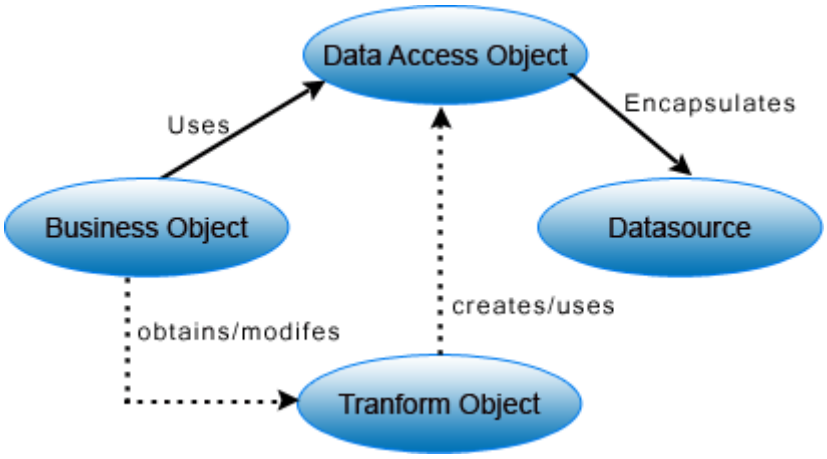
Darba mērķis: Datu piekļuves objektu pētīšana un salīdzināšana ar tiešo datu piekļuvi.

Darba uzdevumi:

1. Noskaidrot datu piekļuves objektu priekšrocības un trūkumus;
2. Noskaidrot tiešās datu piekļuves priekšrocības un trūkumus;
3. Izveidot vienkāršus programmas piemērus izmantojot abas datu piekļuves metodes;
4. Izmainīt datubāzu struktūru un pielāgot abas programmas darbam ar izmainīto datubāzi.

1. DATU PIEKĻUVES OBJEKTI

Datu piekļuves objekti ir kā starpnieks starp programmatūru un datubāzi, kas programmatūrai piegādā nepieciešamos datus kā gatavu objektu, kuram mainīgo vērtības jau ir piešķirtas nevis kā SQL vaicājuma rezultātu kopu. Tiek izveidota atsevišķa klašu kopa, kurai būs izveidotas metodes datu nolasīšanai, saglabāšanai, dzēšanai un rediģēšanai attiecīgajās datubāzēs. Kad programmatūrai nepieciešams sazināties ar datubāzi, piemēram, datu nolasīšanai, tajā ir jāizveido attiecīgais objekts un ar datu piekļuves objekta palīdzību jāsaglabā dati izveidotajā objektā. Tādā veidā datubāzes struktūra nav atkarīga no programmatūras un otrādi (skatīt 1.1. attēlu).



1.1. att. **Datu piekļuves objektu darbības shēma [1]**

Tie ir ļoti piemēroti dažādu objektorientētu programmēšanas valodu, piemēram, JAVA, C#, programmatūras izstrādē, kurām ir nepieciešama sasaiste ar datubāzēm. Būs ērti uzreiz iegūt gatavu objektu no datubāzes datiem nevis iegūt SQL rezultātu tabulu, kura pēc tam vēl jāapstrādā, lai pareizi saglabātu datus programmatūras pareizai darbībai. [1]

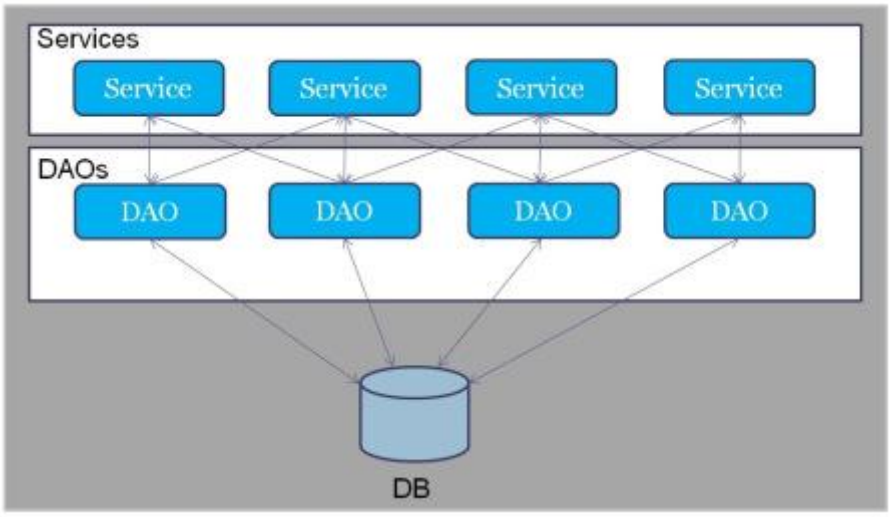
Vairākumam no datu piekļuves objektiem ir raksturīgas vismaz četras metodes, bet bieži tiek izveidotas papildus metodes atkarībā no veicamā uzdevuma, piemēram, datu meklēšana pēc kāda parametra:

- insert() – jauna datubāzes ieraksta izveidei no dotā objekta;
- update() – informācijas atjaunošanai datubāzē;
- delete() – informācijas dzēšanai no datubāzes;
- findAll() – visu attiecīgo objektu atlasīšanai no datubāzes.[2]

1.1. Datu piekļuves objektu priekšrocības

Datu piekļuves objekti nodrošina, ka pašas programmatūras izstrādē, programmētājiem nav jāsatraucas par datubāzes struktūru datu iegūšanai. Ja ir tāda vajadzība izmainīt datubāzes struktūru, piemēram, jaunas kolonas pievienošanu vai kādas tabulas sadalīšanu, tad nav jāsatraucas par pašas programmas koda mainīšanu, kas varētu būt daudz vairāk nekā vienā vietā. Pietiek pareizi pielāgot attiecīgos datu piekļuves objektus jaunajai datu bāzei (papildināt vai izmainīt attiecīgos datubāzu vaicājumus). [5]

Gadās arī tā, ka ar vienu datubāzi darbojas vairākas programmas. Tādā gadījumā, izmantot datu piekļuves objektus ir ļoti izdevīgi, jo nav īpaši jāiespringst veidojot datubāzes struktūru, lai atvieglotu programmatūru izstrādi. Vienkāršāk būtu izstrādāt datu piekļuves objektus katrai programmatūrai un datubāzi varētu izveidot tā, ka būtu ērtāk to pārskatīt, uzturēt vai arī izmaiņu gadījumā nebūtu jāpārraksta liela daļas programmatūras datubāzu vaicājumu. Ir arī iespējams, ka šīm programmām var tikt izmantoti vienu un tie paši datu piekļuves objekti un datubāzes izmaiņu gadījumā būtu jāizmaina tikai datu piekļuves objekts (skatīt 1.2. attēlu). [3]



1.2. att. **Vairākas programmatūras iegūst datus no vienas datubāzes [3]**

Piemēram, tiek izveidota iedzīvotāju reģistra datubāze, kas glabā informāciju par valsts iedzīvotājiem. To izmanto dažādās valsts iestāžu programmas, piemēram, pabalstu aprēķina un pensiju aprēķina programmas. Pastāv pat ļoti liela iespēja izmantot vienu un to pašu datu piekļuves objektu abām programmām.

Programmatūras pirmkods kļūtu saprotamāks, jo tam pa vidu nebūtu dažādi datubāzu vaicājumi. Lai to labāk realizētu, tad noteikti datu piekļuves objektu izstrādē jāievēro noteikta nosaukumu piešķiršanas kārtība, lai pēc nosaukuma programmētājs saprastu, kas tiek iegūts no koda izpildes. Papildus tam, nebūtu arī visos failos jāiekļauj informācija par savienojumu ar datubāzi.

Izmantojot datu piekļuves objektus, uzlabotos arī programmatūras drošība. Galvenā drošības prioritāte būtu tieši datu piekļuves objektu klašu papildus šifrēšana, kas jau lielā mērā palielina drošības līmeni, jo tad, ja arī kādam izdodas tikt pie programmatūras pirmkoda, tad visi savienojuma ar datubāzi parametri viņiem nebūtu pieejami. Tas atvieglotu pirmkoda drošības integrāciju, jo to var ļoti uzlabot nošifrējot tikai dažus failus ne visu pirmkodu.[3]

1.2. Datu piekļuves objektu trūkumi

Lai gan salīdzinoši lielā datubāzes un programmatūras neatkarība vienai no otras ir ļoti liela priekšrocība, pareiza datu piekļuves objektu izmantošana var prasīt vairāk laika tieši plānošanā un atsevišķo klašu izveidē. Tas varētu būt viens no iemesliem, kāpēc datu piekļuves objektus neizmanto ļoti plaši, jo bieži svarīgi ir programmatūru uztaisīt salīdzinoši īsā laika posmā, bet papildus plānošana un datu piekļuves objektu izveide to tikai aizkavēs. Ir ļoti grūti iepriekš līdz galam visu paredzēt, kā programmatūras izstrāde izvērtīsies un kas tās darbībā būs jāmaina.

Lielākais trūkums ļoti veikt spējīgas aparatūras prasīgai programmatūrai ir tas, ka datu piekļuves objekti noteikti izmantos vairāk atmiņas nekā tiešās piekļuves metode. Tas notiek tāpēc, ka atmiņā ir jāsaglabā viss objekts, kaut programmatūras darbībai nepieciešams ir tikai viens mainīgais, kas glabā kādu simbolu virkni vai skaitli. Piemēram, ar sportu saistītās programmās jāsaglabā daudz datu ar spēļu un spēlētāju statistiku. Lietotājs vēlas apskatīt tikai vienu noteiktu spēlētāja statistikas vērtību, piemēram, pārtvertās bumbas, bet, lai iegūtu šo vērtību, vispirms ir jāiegūst paša spēlētāja objekts, kas glabā visus statistikas datus par viņu, un tikai tad var atsevišķi iegūt vienu mainīgo.

Izmantojot datu piekļuves objektus ir iespējams, ka būs vai nu jāizmanto programmatūras izstrādē arī tiešās piekļuves metode, vai nu jāplāno datu piekļuves objektiem papildus mainīgie dažādu papildus vērtību glabāšanai. Tas attiecīgi palielina programmatūras darbībai nepieciešamās atmiņas apjomu un līdz ar to minimālās

aparatūras prasības. Piemēram, ja ir nepieciešams iegūt skaitu, cik daudz spēlētāju ir iemetuši vairāk par 10 punktiem spēlē. Izmantojot tiešo piekļuvi datubāzēm, tas būtu vienkārši iegūstams rezultāts izpildot datubāzu COUNT vaicājumu, bet izmantojot datu piekļuves objektus, vai nu tas speciāli jāiekļauj objekta klasē kā metode, vai nu programmatūras pirmkodā jāmeklē ar cikla palīdzību, ejot cauri visiem datu piekļuves objektiem.

Papildus laiku strukturēšanai un izstrādei vajadzēs, ja būs liela datubāze, no kuras objekti tiek iegūti, jo datu piekļuves objektiem ir raksturīga metode getAllObjects(), kas atgriež visu objektu sarakstu. Saraksts aizņemtu ļoti daudz atmiņas, atgriežot 10 000 datubāzu ierakstu sarakstu. Tas ir ļoti liels trūkums datu piekļuves objektiem, jo, lai to apietu, ir vajadzīgs vai nu izmantot citu datu iegūšanas metodi, kas atkal padara pirmkodu haotiskāku, vai nu jāpārstrukturē pats datu piekļuves objekts, kas var prasīt ļoti daudz papildus laika. [5;6]

1.3. Datu piekļuves objektu ģenerēšana

Pastāv arī dažādi datu piekļuves objektu ģenerēšanas rīki, kas ļauj ģenerēt objektu pirmkodu jau no esošas datubāzes. Lielākoties to darbības princips ir ļoti vienkāršs. Tiem jāpadod datubāzes, attiecīgo tabulu un kolonu nosaukumi, kuras vērtības vēlaties iekļaut datu piekļuves objektā. Ir arī tādi ģeneratori, kas savienojas ar datubāzi un ļauj grafiskā vidē izvēlēties datu piekļuves objektam nepieciešamās kolonas un izveidot tiem nosaukumus.

1.3.1. DaoGen

Viens no datu piekļuves objektu ģeneratoriem ir DaoGen, kas ļauj ģenerēt datu piekļuves objektu pirmkodu, norādot tabulas nosaukumu, programmēšanas valodu, datubāzes servera tipu, kolonu skaitu un katras kolonas tipu un nosaukumu. Pareizi ievadot datus ieguvu jau gatavu pirmkodu datu piekļuves objektam. Tas viss notiek tiešsaistē. pirmkods pēc tam ir jāiekopē attiecīgajā programmā, ar kuru pirmkods tiek rediģēts. Izmantojot savu mazo datubāzes struktūru mēģināju uzģenerēt datu piekļuves objektu ar šo rīku. Tas ļoti veiksmīgi izdevās, papildus paša objekta izveides kodam ieguvu arī SQL tabulas izveides pirmkodu, ko gan šoreiz nevajadzēja.

Tā kā šo rīku es izmēģināju pirmo, tad uzreiz jau saskatu vairākus nozīmīgus trūkumus:

1. Nav iespējams uzģenerēt datu piekļuves objektu, kas iegūtu datus no vairākām tabulām;
2. Tiek izveidotas tikai noteiktas metodes, lai tālāk strādātu ar datubāzi, kas nozīmē, ka kods tāpat būs jāpapildina ar nepieciešamām metodēm;
3. Kā jau lielākā daļa ģenerētā koda, no tā noteikti kaut kas būs arī jāizdzēš, jo nesakritīs ar programmatūras izstrādes principiem vai arī vienkārši nebūs nepieciešams.

Ja nav nepieciešams sasaistīt vairākas tabulas vienā datu piekļuves objektā un ir nepieciešamas tikai vienkāršākās metodes darbā ar tiem, tad DaoGen ir labs rīks ar ko var ātri un ērti izveidot datu piekļuves objektus datubāzēm. [4]

1.3.2. DAO-Generator

Atšķirībā no DaoGen rīka, šis rīks jau ir lejupielādējams un darbināms uz paša datora. Tas ir pieejams tiešsaistē http://dao-data-access-object-generator.soft32.com/. Ir pieejama gan versija, kas domāta komerciālām darbībām, gan tāda, kurai ir ierobežota funkcionalitāte.

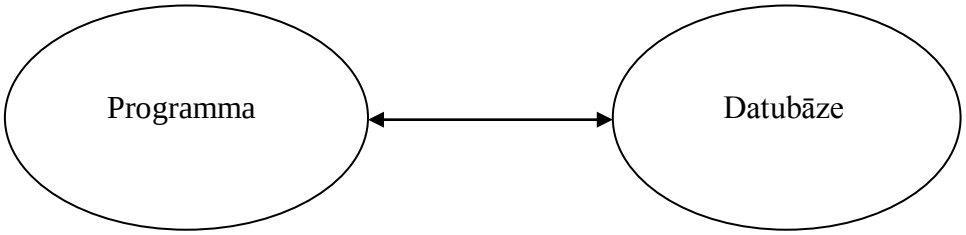
Izmantojot to pašu nelielo datubāzi, mēģināju izveidot datu piekļuves objektus ar šo rīku. Programma ir rakstīta JAVA valodā un izmanto tās bibliotēkas, lai izveidotu savienojumus ar datubāzēm, tāpēc bija nepieciešams nedaudz laika, lai pareizi nokonfigurētu programmu manām vajadzībām.

Tā kā sagatavošanās datu piekļuves objektu ģenerēšanai jau prasīja vairāk laika un piepūles, tas tikai liecina par rīka plašajām iespējām. Rīks savienojas ar manis izveidoto datubāzi. Pēc tam ir iespējams norādīt tabulas un kolonas, no kurām dati ir jāiegūst. Ir iespējams arī izveidot datu piekļuves objektus no vairākām datubāzēm.

Rezultātā tiek iegūta mape ar vairākām apakšmapēm, kurās atrodas dažādi java faili, kas satur gan pašu datu piekļuves objektu, gan tā realizāciju. Uzreiz ir redzams, ka šim rīkam ir daudz plašākas iespējas salīdzinājumā ar DaoGen rīku. Rīku lietderīgi iespējams izmantot jau daudz lielākiem un sarežģītākiem projektiem. Kā jau visiem ģenerētajiem kodiem, noteikti ir uzģenerēts kaut kas lieks, kas nemaz nav nepieciešams, vai arī kaut kas tieši pietrūkst, bet es domāju, ka jebkuram šāda veidā ģenerētam kodam būs nepieciešamas korekcijas.

2. DATU TIEŠĀ PIEKĻUVE

Datu tiešā piekļuves metode ir tad, kad datubāzu vaicājumus raksta tieši programmatūras pirmkodā. Ja ir nepieciešami dati no datubāzes, tad pirmkodā tiek izveidots savienojums ar datubāzi, izveidots datubāzes vaicājums, no kura tiek iegūta rezultāta kopa. Pēc tam no tās ir atsevišķi jāiegūst vērtības un jāsaglabā tās attiecīgajos mainīgajos. (skatīt 2.1. attēlu)



2.1. att. **Datu tiešās piekļuves shēma**

2.1. Datu tiešās piekļuves priekšrocības

Pati lielākā datu tiešās piekļuves priekšrocība salīdzinājumā ar datu piekļuves objektiem ir ātrāka programmatūras izstrāde. Lai iegūtu datus no datubāzes ar tiešo piekļuvi, programmatūras pirmkodā ir jāraksta datubāzu vaicājumi, kas kā rezultātu atgriež kopu ar nepieciešamajām vērtībām. Tā kā to nākas realizēt tikai tur, kur programmas izpildē tas ir nepieciešams, nevis pirms tam sagatavotā objektā, tad nav jāveic gandrīz nekāda papildus plānošana pirms programmēšanas. Tādejādi samazinot programmatūras izstrādes ilgumu, kas mūsdienu programmu izstrādē ir ļoti nozīmīgs faktors.

Par otru lielāko datu tiešās piekļuves priekšrocību es uzskatu to, ka tā izmanto tikai tik daudz atmiņas, cik nepieciešams noteiktajam uzdevumam. Tā kā mūsdienās aparatūra ir ļoti attīstījusies, daudzās programmatūras izstrādēs izmantotais atmiņas daudzums var šķist nebūtisks. It sevišķi, ja starpība starp vienas un otras metodes izmantošanu nemaz nav tik liela. Bet tas liekas nebūtiski tikai pie mazākām datubāzēm, piemēram, ja datubāzē ir 10 000 ierakstu, no kuriem varētu izveidot 10 000 datu piekļuves objektu, tad, lai noskaidrotu ierakstu skaitu ar kādu specifisku parametru, izmantojot datu piekļuves objektus, teorētiski būtu atmiņā jāglabā 10 000 objektu. Kas šajā ziņā gan būtu ļoti nozīmīgi. [7]

2.2. Datu tiešās piekļuves trūkumi

Programmatūras pirmkods ir daudz „piesārņotāks" ar dažādiem datubāzu vaicājumiem, kas apgrūtina tā lasāmību. Protams ir iespējams, visus datubāzu vaicājumus salikt atsevišķā failā, bet tas nemainīs situācija, ka pirmkodā tāpat būs jāveic datubāzu vaicājumu parametru piešķiršana un izpilde.

Tā kā praktiski visas programmas izstrādes gaitā tiek mainītas vai uzlabota tās funkcionalitāte, tad gandrīz neizbēgama ir datubāzes struktūras izmainīšana. Ja tiek veiktas datubāzes struktūras izmaiņas, tad programmas pirmkodā visi ar izmaiņām saistītie datubāzes vaicājumi ir jāizlabo, lai strādātu pareizi ar jaunajām izmaiņām. Tas ļoti lielā mērā nozīmē ilgu meklēšanu pa pirmkodu, kur viss ir jāatjauno. Mazākās programmās, tā gan nav tik būtiska problēma, bet programmās ar plašu funkcionalitāti un datubāzu vaicājumu dažādību – šādas izmaiņas var ļoti pagarināt izstrādes ilgumu. [7]

2.3. Secinājumi

No abu metožu teorētiskās daļas varu izsecināt, ka galvenie kritēriji datu piekļuves metodes izvēlē ir šādi:

- Programmas izveides ātrums (gan plānošana, gan izstrāde);
- Programmas rediģēšanas ērtums un ātrums;
- Izmantojamā atmiņa programmas darbības laikā;
- Servera lieka noslogošana.

Izpētot vairākus teorijas avotus, varu viegli izsecināt, ka izstrādājot programmas ar tiešo datu pieeju, tās izstrādes laiks salīdzinājumā ar datu piekļuves objektiem ir mazāks. Līdz ar to ļoti efektīva metode, ja programmatūras izstrādei atvēlēts maz laika.

Kategorijā par programmatūras koda rediģēšanu viennozīmīgi uzvar datu piekļuves objekti. Tas ir tāpēc, ja, piemēram, ir jāizmaina datubāzes struktūra, bet izmaiņām nevajadzētu iespaidot programmatūras funkcionalitāti (programmai nav jāprot izpildīt jaunas funkcijas vai arī kāda tieši ir jālikvidē). Tad viennozīmīgi datu piekļuves objektiem ir liela priekšrocība, jo pietiktu pielāgot tikai paša objekta realizāciju, nevis programmatūras kodu.

Varu droši izsecināt, ka datu piekļuves objekti noteikti izmanto vairāk atmiņas nekā, ja datus iegūtu pa tiešo no datubāzes. Datu piekļuves objektiem ir jābūt metodei getAll(), kas atgriež visu objektu sarakstu un tas noteikti izmanto daudz atmiņas. Tāpēc, ja vajadzīgs atmiņas taupošs risinājums, labāk izvēlēties datu tieši piekļuvi.

Toties, ja runa ir par servera noslogošanu, tad datu piekļuves objektiem šeit jāuzrāda ļoti labi rezultāti. Tie saziņu ar datubāzi veic tikai tad, kad nepieciešams, jo patiesībā ir kā maza datubāze programmas atmiņā, no kuriem var iegūt nepieciešamo informāciju datu apstrādei (piemēram, kāds vaicājums ar COUNT() lauku, nav jāprasa datubāzei, bet gan var vienkārši ar cikla palīdzību pats manuāli saskaitīt nepieciešamās vērtības). tas gan palielina programmatūras izstrādes sarežģītību un ilgumu.

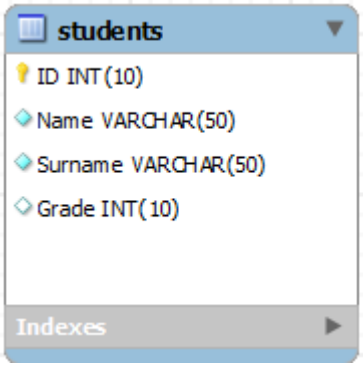
3. PROGRAMMU IZVEIDE UN ANALĪZE

Teorētiskais materiāls ir pietiekami apskatīts un izpētīts, lai gūtu plašāku priekšstatu par datu piekļuves objektu priekšrocībām, trūkumiem un īpatnībām. Ir nepieciešams pārbaudīt teoriju praksē, izveidojot vienkāršas programmas darbam ar SQL datubāzi. Kopumā jāizveido un jāstrādā ir ar diviem programmas variantiem. Viens, kas izmanto datu piekļuves objektus datu saņemšanai no datubāzes, bet otrs, kas izmanto datu tiešo piekļuvi. Lai spētu kvalitatīvāk salīdzināt, kura metode ir ērtāka, izdevīgāka vai vienkāršāk saprotama tieši programmētājiem, abu programmu darbībām un iespējām jābūt pēc iespējas vienādām. Programma spēj veikt vienkāršākās darbības ar datubāzes datiem (izveidot, dzēst un meklēt ierakstus). Abas programmas tiek izstrādātas valodā JAVA. Pēc abu programmu izstrādes, tiks izmainīta datubāzes struktūra. Tā kā tagad tas ir tikai viena kontroldarba atzīme, pieņemsim, ka tas norāda, ka tikai viens pasniedzējs glabāja vērtējumus datubāzē. Citi pasniedzēji ievēroja, ka tā pašiem nav jākrāmējas ar tik daudz papīriem, tāpēc arī sagribēja ieviest sev šādu iespēju.

Tāpēc datubāze tiks pārstrukturizēta tā, ka sastāvēs jau no trīs atsevišķām tabulām. Tabulā par studentiem glabāsies to ID numurs, vārds, uzvārds un e-pasts, tātad pazudīs kolonna ar vērtējumiem, bet parādīsies kolonna ar studentu e-pastiem, ja tādi būs norādīti. Tiks izveidotas divas papildus tabulas. Pirmā saturēs tādu priekšmeta informāciju kā priekšmeta ID, nosaukums un pasniedzējs. Otrā tabulā tiks glabātas tikai attiecīgā studenta atzīmes norādītajos priekšmetos (studenta ID, priekšmeta ID, atzīme).

3.1. Programmas izveide

Sākumā datubāze sastāv no vienas tabulas, kurā glabājas visu studentu vārdi, uzvārdi un iegūtās kontroldarba atzīme vienā priekšmetā. Tā kā teorētiski datu piekļuves objekti esot izdevīgāki par tiešās piekļuves metodi, ja ir jāveic izmaiņas datubāzē, tad darba gaitā veikšu datubāzes izmaiņas, pievienojot papildus laukus un tabulas, kā arī izņemot kādu kolonu no studentu tabulas (skat.3.1.attēlu).



3.1. att. **Sākotnējā datubāzes struktūra**

3.1.1. Datu piekļuves objekta izveide

Pirmo sāku izstrādāt vienkāršu programmu, kas izmanto datu piekļuves objektu, lai iegūtu datus no datubāzes. Protams, pirms pašas programmas izveides bija nepieciešama nedaudz lielāka plānošana, jo testa programma izstrādes procesā sastāvēja no četriem failiem (klases interfeisa, studenta datu klases, datu piekļuves objekta realizācija un pati programmas klase).

Izveidoju datu piekļuves objekta interfeisu ar tam nepieciešamajām metodēm (getAll, update, delete, create). Tālāk ķeros pie paša objekta realizācijas. Sākumā līdz galam nebiju sapratis datu piekļuves objektu būtību un sāku jau domāt, kādas papildus metodes man vajadzētu, lai attiecīgos uzdevumus realizētu. Izveidoju realizāciju ar dažām liekām metodēm. Sakot izstrādāt pašu testa programmu, kas darbosies ar datu piekļuves objektu, es sapratu, kur esmu nedaudz pārpratis būtību un nāksies nedaudz objekta realizāciju izlabot. Kā izrādās, lielāko daļu uzdevumu var paveikt izmantojot metodi getAll(), jo ar to iegūst sarakstu ar visiem objektiem. Pēc tam jau viss ir atkarīgs no programmētāja. Visus vajadzīgo informāciju var atlasīt programmas pirmkoda daļā. No šī incidenta, izrietēja mans pirmais secinājums, ka datu piekļuves objektu ieviešana programmā ir ļoti pamatīgi jāizvērtē, vai to izmantošana atmaksāsies.

Kad veiksmīgi biju pārvarējis savu kļūdu un pārveidojis datu piekļuves objekta realizāciju, atkal varēju atsākt izstrādāt testa programmu. Testa programma ir ļoti vienkārša – tā prasa lietotājam izvēlēties no piedāvātajām darbībām un attiecīgi dara, ko lietotājs vēlas. Bet kā jau vairākkārtīgi tika pieminēts, tad datu piekļuves objekti prasa daudz vairāk plānošanas un spēju saskatīt iespējamās problēmas agrāk, lai tās būtu vieglāk atrisināt. Tā lūk, es saskāros ar problēmu, ka, ja datubāzē, kurā kādā tabulā primārajai atslēgai vērtības tiek automātiski piešķirtas pie datu saglabāšanas datubāzē, tad šī primārā atslēga ir jāiekļauj arī datu piekļuves objektu sarakstā pie jaunizveidotā objekta. Kādu laiku mēģināju apsvērt dažādas iespējas, kā to realizēt – varbūt noņemt datubāzē primārās atslēgas vērtības piešķiršanu no AUTO_INCREMENT, bet gan to kontrolēt no programmas. Tā gan nelikās laba ideja, jo tad varētu rasties problēmas ar primārās atslēgas dublēšanos. Pastāv arī iespēja izmantot datubāzu vaicājumu „SELECT LAST_INSERT_ID();”, kas kā rezultātu kopu atgriež pēdējo automātisko vērtību, kas ar INSERT vaicājumu ir tikusi ievietota. Izmantojot šo vaicājumu gan var rasties problēmas, ja tiek atļauts darboties vairākām aplikācijām vienlaicīgi ar vienu datubāzi un neizsaucot šo vaicājumu uzreiz pēc vajadzīgā INSERT vaicājuma, kā rezultātu var iegūt kāda INSERT vaicājuma automātisko vērtību. Manā gadījumā, tas gan neko neiespaido, jo programma tiek izveidota mācību nolūkos, kas dziļāk nepēta šādas situācijas, tāpēc man pietiks ar minēto SELECT vaicājuma izpildi uzreiz pēc datu saglabāšana datubāzē. Pēc tam pareizi apstrādāju rezultāta kopu, ieguvu ievietotā ID vērtību, ko pēc tam varēju saglabāt programmas sarakstā attiecīgajam objektam.

3.1.2. Datu tiešās piekļuves izveide

Salīdzinājumā ar datu piekļuves objektu izveidi un struktūru, datu tiešajā piekļuvē es uzreiz sāku rakstīt testa programmas pirmkodu un lieki nesatraucos ne par kādām problēmām. Es visus datubāzu vaicājumus glabāju atsevišķos mainīgajos programmas pirmkoda sākumā, jo, iespējams, būs vajadzība kādu vaicājumu izpildīt atkārtoti vairākas reizes. Papildus tam, ievērojot šādu struktūru, vismaz pirmkods izskatītos daudz sakārtotāks un izmaiņu gadījumā datubāzu vaicājumi atrastos katrā failā tikai vienā vietā. Protams, datu apstrāde tāpat būtu jāmeklē atsevišķi un jāpielāgo veiktajām izmaiņām, lai programma spētu pareizi funkcionēt. Programmas darbības princips ir identisks tai programmai, kas tika realizēta ar datu piekļuves objektiem. Ar papildus grūtībām vai neizpratni programmas izstrādes laikā nesaskāros.

Izmantojot datu tiešo piekļuvi, programmu bija iespējams izstrādāt samērā ātri un bez liekām problēmām, kā tas bija ar datu piekļuves objektu programmu. Izstrādes gaitā gan jutu, ka daudz kur kods ļoti līdzinās citur izmantotajam un varētu strukturizēt programmu efektīvāk.

3.1.3. Secinājumi

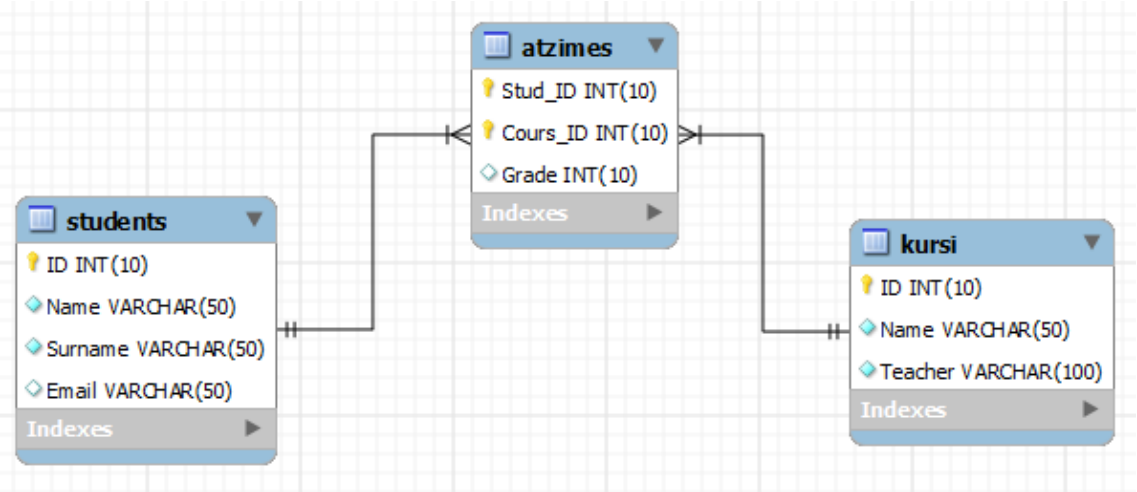
Pašam izstrādājot abas, programmas varēja ļoti labi sajust, ka abām metodēm ir nedaudz raksturīgs savs programmēšanas stils. Datu piekļuves objektiem tas ir šķietami koptāks, jo tiek izmantota koda sadalīšana sīkāk, kas dod labāku koda pārskatu un struktūru. Taču, bija arī ļoti jūtams, ka izmantojot datu pieejas objektus, ir ļoti jāpiedomā pie tā, kas īsti ir nepieciešams programmas pareizai funkcionēšanai.

Toties izmantojot datu tiešo piekļuvi, ir jūtama lielāk brīvība rakstot programmas kodu, kaut vēlams vismaz datubāzu vaicājumus glabāt vienā vietā, lai pēc tam tie nav jāmeklē pa visu kodu. No otras puses atkal, kods paliek daudz „piesārņotāks" un grūtāk salasāms. Šo aspektu ļoti daudz nepalīdz uzlabot paskaidrojošu mainīgo nosaukumu lietošana, jo programmas kodā parādās pārāk daudz dažādu ar datubāzēm saistītu metožu nosaukumi, ka ļauj vieglāk apmaldīties kodā, jo liela daļa no tā izskatās ļoti līdzīgi.

Arī teorētiskajā daļā minētais programmas izveides ilgums, ļoti lielā mērā atbilda manis veidotajai programmai. Respektīvi, programmu, kas izmanto datu piekļuves objektus, nācās gan pirms programmēšanas, gan tās laikā saplānot, lai būtu ērtāk un ātrāk realizēt. Taču ar tiešo datu pieeju – tādu problēmu nav, jo visi dati, kuri ir nepieciešami programmai, glabājas datubāzē un lai tos iegūtu vajag tikai izveidot pareizu datubāzes vaicājumu un attiecīgi apstrādāt rezultāta kopu.

3.2. Izmaiņas datubāzē un programmā

Kad esmu veiksmīgi ticis galā ar abu programmu izveidi un veicis dažus secinājumus un testus par programmu vienādu rezultātu sniegšanu, ir pienācis laiks veikt datubāzes struktūras izmaiņas. Tādā veidā spēšu precīzāk noskaidrot, cik daudz pūļu un laika prasīs abu programmu pielāgošana jaunajai datubāzei (skatīt 3.2 attēlu).



3.2. att. **Datubāzes struktūra pēc izmaiņām**

Tā kā izmaiņas datubāzes struktūrā ir ļoti krasas, tad uzreiz ir skaidrs, ka abu programmu pirmkoda pielāgošana prasīs diezgan daudz laika un pūļu, jo klāt nāks daudzas funkcionalitātes pārbaudošas vajadzības. Piemēram, izvadīt visus kursus, ko apmeklē kāds students, vidējā atzīme utt. Galvenā prasība abām programmām ir iekļaut šādu funkcionalitāti:

- Iespēja apskatīt visu studentu atzīmes, atsevišķi parādot katram studentam attiecīgā kursa atzīmi;
- Rediģēt studenta personas un mācību datus;
- Dzēst visa veida datus;
- Apskatīt visu informāciju tikai par vienu studentu.
- Izveidot jaunu ierakstu datubāzē par jaunu studentu.

3.2.1. Datu piekļuves objekti

Uzreiz jau ienāca prātā ideja par otra datu piekļuves objekta izveidi, kas saturētu informāciju par kursiem. Tā kā veidojot datu piekļuves objektu par studentiem nācās saskarties ar pailgu plānošanu un negaidītiem šķēršļiem, tad šoreiz darīšu tā, ka katram studentam jau esošajā objektā būs saraksts ar kursiem, ko tas apmeklē un atzīmēm. Tā varētu būt mazāka iespēja, ka rastos kādas problēmas, par kurām es nebūtu jau pirms koda rakstīšanas padomājis.

Protams, veidojot programmu strukturētāku, nākas izveidot jaunu klasi, kas satur trīs mainīgos (kursa nosaukumu, pasniedzēju un atzīmi). Tagad no studentu klases jāizdzēš mainīgais, kas saturēja vienu atzīmi un jāaizstāj tas ar kursu sarakstu, ko apmeklē students. Kā jau pierasts darbā ar datu piekļuves objektiem, tad uzreiz izsecinu,

ka šādu izmaiņu gadījumā, programmas darbības laikā nepieciešamās atmiņas daudzums krasi pieaugs.

Abas iepriekš minētās izmaiņas bija sīkums, salīdzinājumā ar to kas vēl jādara. Nākamais solis pretim pareizi strādājošai programmai ir paša datu piekļuves objekta realizācijas pielāgošana jaunajai datubāzes struktūrai. Tieši šajā vietā atkal ir jācenšas ieplānot veicamos darbus uz priekšu un paredzēt iespējamās problēmas un to risinājumus, lai vēlāk nebūtu jāatgriežas pie šīs klases labošanas, jo tas var novest pie ietilpīgas testa programmas pirmkoda labošanas.

Kārtīgi izvērtēju katras studentu datu piekļuves objekta metodes, lai saprastu, ar kurām metodēm labāk sākt pielāgot objektu. Ātri sarakstot idejas, kas būtu jāmaina katrai metodei, nonācu pie slēdziena, ka būs jāsāk ar pašu vienkāršāko un ātrāk izdarāmo metodi. Respektīvi ar dzēšanu, jo, lai programma pareizi strādātu, tai ir tikai pirms paša studenta datu dzēšanas jāizdzēš dati par studenta atzīmēm.

Nākošais sarakstā ir paša objekta konstruktors, jo tajā tiek izveidots visu objektu saraksts, kam arī piešķir vērtības no datubāzes. Katra studenta datu piekļuves objektam papildus jāizveido saraksts ar mācību kursiem, kuros tam ir vērtējums. Pagaidām arī šis uzdevums liekas salīdzinoši vienkāršs, jo atlika tikai izveidot katram objektam papildus datubāzes vaicājumu, kas atlasa pareizos datus par studenta vērtējumiem un tos pareizi saglabā sarakstos.

No sākuma liekas, ka arī datu piekļuves objekta izveides metodes pielāgošana pārmaiņā varētu nebūt nekas sarežģīts. Taču pārrakstot metodes kodu, saprotu, ka būs jāveido papildus nosacījumi. Pirmkārt, vai atzīmju saraksts ir tukšs vai nav. Otrkārt, jāiegūst attiecīgā kursa ID numurs, kas jānoskaidro no tā nosaukuma un pasniedzēja. To varētu risināt divējādi. Vispareizākais būtu pievienot kursa klasei mainīgo, kas saturēs šo ID numuru. Taču es pieņemu, ka šīs programmas datubāzē nebūs tādu ierakstu, kas radītu sistēmas kļūmi. Varbūt ar to pats sev apgrūtināju izmaiņu ieviešanu, jo būs jāveido sarežģītāks datubāzes vaicājums, lai iegūtu pareizo informāciju. Tā kā šī programma tiek veidota mācību nolūkos, tad tas arī nenāks par sliktu.

Kā pēdējai klāt ķēros update() tipa metodei, kura jau sākumā likās, ka būs visgrūtāk pielāgojam izmaiņām, jo klāt nācis vēl viens saraksts ar vērtībām, kuram jāiet cauri un jāpārbauda, vai visas vērtības jau ir datubāzē vai nav. Arī šajā metodē pastāvēja iepriekš minēta problēma par kursa ID numuru, kas tika atrisināta tieši tāpat kā ar jauna

ieraksta izveidi, proti, jāiegūst ID no nosaukuma un pasniedzēja. Tomēr lielākās grūtības sagādāja vērtējumu pielāgošana. Respektīvi, katram studentam ir vairāki kursi, kuros viņam ir atzīme, bet ir arī tādi, kuros nav atzīmes. Tātad šeit jāveido divi atsevišķi datubāzes vaicājumi. Viens kursiem un atzīmēm, kas tiek labotas, bet jau atrodas datubāzē. Otrs kursiem un atzīmēm, kurām datubāzē atzīmes nav saglabātas. To visu salikt kopā bija nedaudz piņķerīgi, bet viss sanāca. Ar to arī beidzas paša datu piekļuves objekta pielāgošana datubāzes izmaiņām. Tā kā datubāzes struktūra mainījās ļoti krasi no tās, kas bija sākumā, tad es paredzu, ka pašu testa programmu arī nāksies ilgāku laiku pielāgot – vispirms izlabot radušās kļūdas un tad papildināt ar testa iespējām jaunajai funkcionalitātei. Izmainītā datu piekļuves objekta realizācija ir apskatāma 1. pielikumā.

Iesākumam nākas izdzēst neesošo metodes izsaukumus (getGrade()/setGrade()), kas traucē programmas pirmkoda kompilēšanu un darbināšanu. Nākas kārtīgi arī pārdomāt, kādus datus būtu vērts attēlot un ko ļaut ar tiem lietotājam darīt. Tā kā galvenais testa programmas uzdevums ir darbs ar datu piekļuves objektiem, tad programmā noteikti jāiekļauj darbības, kas ļautu izmanto katru no objekta metodēm.

Pirmkārt, sāku ar datu galveno datu izvadi saistībā ar manis izveidoto datu piekļuves objektu. Izvadu studenta vārdu, uzvārdu un visas viņa atzīmes, ja tādas ir. Šajā ziņā daudz nekas no iepriekšējās programmas nebija jāpielāgo, tikai jāpievieno cikls, kas izvada visus studenta kursa saraksta vērtējumus.

Sarežģītākā daļa šajā posmā ir datu rediģēšanas iespējas nodrošināšana, kas datubāzes struktūras izmaiņas dēļ, ir jāmaina gandrīz pilnībā. Manuprāt, rediģēšanai jāizdala divās daļās:

- Personas datu rediģēšana, kurā varēs nomainīt studenta vārdu, uzvārdu vai e-pastu.
- Mācību datu rediģēšana, kurā varēs ielikt studentam atzīmi par kāda kursa nokārtošanu.

Tā kā salīdzinoši daudz laika un pūļu ieliku datu piekļuves objekta izveidē, tad šāda funkcionalitātes sadalīšanā nevajadzētu rasties papildus problēmām. Tieši tā arī izdevās, jo viss, kas bija jāizlabo, bija jānomaina atzīmes mainīgais uz e-pastu, jo tagad pie studenta datiem vairs neglabājas atzīme, bet gan studenta e-pasta adrese.

Turpinot iesākto darbu, kārtējo reizi saskāros ar neparedzētu problēmu. Lai ieliktu kādam studentam atzīmi priekšmetā, kas nav attiecīgā studenta kursu sarakstā, man nepieciešams iegūt visu kursu datus. Atkal pastāv vairāki varianti, kā to realizēt – izveidot kursu sarakstu tajā pašā datu piekļuves objektā, var to iegūt rakstot datubāzu vaicājumu tieši programmas kodā, bet šis variants šoreiz atkrīt, vai arī pievienot kursa aprakstošajai klasei. Tātad izveidoju tajā pašā datu piekļuves objektā otru sarakstu, kas satur visus pieejamos kursus. Veicot šīs funkcionalitātes nodrošināšanu, varēja izjust, ka datu piekļuves objekti lielu daļu datu apstrādes veic programmas pusē nevis pa tiešo izmantojot datubāzu vaicājumus. Nācās iet cauri izvēlētā studenta kursiem, lai noskaidrotu, vai ir likta jauna atzīme, vai labota jau esoša.

Pārbaudot nākamo funkcionalitātes pareizību manā sarakstā, atklājās, ka studenta datu dzēšana jāmaina programmas kodā nav vispār. Visas izmaiņas tika izdarītas jau datu piekļuves objektā. Tas tikai norāda, ka, ja funkcionalitāte nemainās, tad tiešām jāpamaina tikai datu pieejas objekta realizācija, bet pašu programmas kodu var atstāt neaiztiktu.

Nākamais solis uz pilnīgu pielāgošanu ir nedaudz pamainīt jauna studenta pievienošanu sistēmai. No iepriekšējās programmas versijas ir tikai jānomaina atzīmes vērtības piešķiršana uz e-pasta piešķiršanu.

Atlicis tikai pēdējais posms, lai pabeigtu pielāgošanu jaunajai datubāzei. Atlasīt un parādīt viena studenta datus. Nekādas lielās izmaiņas arī nav jāveic, tikai jāpievieno papildus dati, ko izvadīt uz ekrāna apskatei. Ar to arī beidzas datu piekļuves objektu pielāgošana jaunajai datubāzei. Programmas pirmkodu skatīt 2. pielikumā.

3.2.2. Datu tiešā piekļuve

Tā kā programmā visi datubāzu vaicājumi ir ievietoti pirmkodā, tad pirmais, kas ir jāizdara, ir jāpārtaisa vienkāršākie datubāzu vaicājumi, lai tie atbilstu jaunajai datubāzes shēmai. Tā kā sākumā bija tikai viena tabula, kurai tagad ir izmainījies tikai viens lauks, tad esošajos datubāzu vaicājumos vienkārši ir jāizvieto vecais kolonas nosaukums ar jauno (Email).

Papildus esošajiem datubāzes vaicājumiem būs nepieciešami jauni, jo funkcionalitāte ir izmainījusies un kļuvusi sarežģītāka. Piemēram, vajadzēs izveidot jaunu datubāzu vaicājumu, kas iegūst visus kursus, kurus apmeklē students.

Lai izvadītu visus nepieciešamos datus uz ekrāna, nepieciešams izveidot jau minēto datubāzes vaicājumu, kas atgriež studenta kursus. Šis vaicājums rezultātā atgriezīs tabulu ar divām kolonnām – Grade un Name, kur Grade ir atzīme un Name ir priekšmeta nosaukums. Tas viss vēl papildus jāievieto ciklā, kas izvada tabulāros datus uz ekrāna. Tās ir galvenās izmaiņās visu datu ieguvei un izvadei uz ekrāna.

Lai cik negaidīt tas būtu, studenta personas datu rediģēšanā, jānomaina bija tikai dažādi mainīgo nosaukumi un datu tipi (no int Grade uz String Email), lai tie atspoguļotu datu saturu. Protams, arī tas pats attiecīgi jāizdara ar datubāzu vaicājumiem, kas ir iesaistīti šajā darbībā.

Programmas daļā, kurā tiek ielikta studentam atzīme kādā priekšmetā, ir jāmaina visvairāk. Tagad papildus tam, ka jānorāda students, kuram atzīme tiek ielikta, jānorāda arī kurss, kurā students saņēmis atzīmi, bet tās ir tikai vieglākās izmaiņas. Kad students un kurss ir izvēlēts, tad jāpārbauda, vai studentam šajā kursā jau ir atzīme. Ja ir tad jāveic UPDATE datubāzes vaicājums, ja nav, tad INSERT datubāzes vaicājums.

Datu dzēšanas sadaļā, kā jau visās, ir jānomaina mainīgo nosaukumi un tipi, lai atbilstu jaunajai funkcionalitātei. Papildus tam, jāizveido datubāzes vaicājums, kas izdzēš visas studenta atzīmes no tabulas „atzīmes". Tas ir nepieciešams, lai varētu izdzēst paša studenta datus no datubāzes, jo šīs tabulas ir saistītas ar ārējo atslēgu. Tas arī viss priekš studentu datu dzēšanas izmaiņām.

Jaunajā datu bāzē ir iespējams, ka vienam studentam ir vairākas atzīmes. Tāpēc pēc studenta datu iegūšanas, jāizveido papildus datubāzes vaicājums, kas atgriež rezultāta kopā visas studenta atzīmes un kursu nosaukumus, kuros atzīmes ir nopelnītas.

Kā jau daudzviet, arī pie iespējas apskatīt tikai viena studenta datus, nepieciešams gan mainīgos pareizi noformēt, gan papildus datubāzes vaicājums, kas iegūtu atzīmes un kursu nosaukumus no datubāzes. Šoreiz gan šāds datubāzes vaicājums jau ir izveidots, jo tas bija nepieciešams arī visu studentu atzīmju un kursu iegūšanai, un izmaiņu apjoms un laika nepieciešamība samazinās.

Pēdējā kategorija ir studenta datu pievienošana (vārds, uzvārds un e-pasts) datubāzei. Šajā daļā izmaiņas atkal ir gaužām vienkāršas. Šoreiz atliek tikai mainīgo pārsaukšana programmas kodā un tabulas kolonnu nosaukumu izlabošana datubāzes vaicājumā. Viss programmas pirmkods atrodams 3. pielikumā.

3.2.3. Secinājumi

Veicot programmas koda pielāgošanu jaunajām prasībām, secināju vairākas lietas, kas atteicas tieši uz programmas izmaiņu veikšanu.

1. Datu piekļuves objektu pielāgošana izmaiņām var novest pie izmaiņu veikšanas gan objekta klases failā, gan programmas kodā. Man sanāca, ka bija, gan reizes, kad vajadzēja mainīt abas, gan kad programmas kods jāatstāj tāds pats kāds bija.
2. Pirmā iemesla dēļ, programmas, kas izmanto datu piekļuves objektus, pielāgošana var būt arī diezgan piņķerīga, jo ir jāsaprot, kurā kodā labāk veikt izmaiņas (programmas vai datu piekļuves objekta).
3. Ļoti krasas izmaiņas programmas funkcionalitātē vieglāk un saprotamāk veikt, ja ir datu piekļuves objekti.
4. Ja izmaiņas iekļauj jaunu datu iegūšanu no citām tabulām, tad vieglāk likās izmantot tiešo pieeju, jo nebija jādomā, kā datus saglabāt.
5. Ja ir jāizmaina tikai kāda mainīgā nosaukums vai tips, tad abām metodēm izmaiņu veikšana ir ļoti vienlīdzīga, gan laika, gan sarežģītības ziņā.
6. Veicot ļoti apjomīgas izmaiņas, kas saistās ar datu struktūru paplašināšanu, izmantojot tiešo pieeju kods ar laiku paliek ar vien grūtāk uztverams. Lai varētu viegli strādāt ar programmu, tad būtu nepieciešama koda struktūras pārskatīšana un uzlabošana.

REZULTĀTI

Darba izstrādes rezultātā tika iegūtas padziļinātas zināšanas par datu piekļuves objektiem, to darbības principiem, priekšrocībām un trūkumiem. Tās izmantojot, tika patstāvīgi izveidotas programmas darbam ar datu bāzi, no kurām viena izmantoja datu piekļuves objektus. Tas deva arī praktiskas iemaņas šādu uzdevumu veikšanai.

Rezultātā tika noteikti iemesli, kāpēc izmantot vai neizmantot datu piekļuves objektus programmas datu apmaiņai ar SQL datubāzi. Pēc teorijas sīkākas pētīšanas, lielākais iemesls izmantot datu piekļuves objektus ir tā vieglā pielāgojamības īpašība, kas ļauj vieglāk programmai pielāgoties pie datubāzes izmaiņām. Taču izstrādājot programmu, kurai jāveic pielāgošanās jaunai datubāzei, brīžiem likās, ka pie noteiktām izmaiņām, tas nenotiek nemaz tik viegli kā gribētos.

Toties teorētiski, datu piekļuves objektu izveide prasa vairāk laika nekā vienkārši rakstīt datubāzu vaicājumus pa tiešo programmas kodā. Šim apgalvojam es varu piekrist, bet par tik laikietilpīgu procesu to pārsvarā padara nepieciešamā plānošana pirms programmas koda izstrādes un kādu neieplānotu problēmu risināšana, kas var prasīt kādu sarežģītāku risinājumu.

SECINĀJUMI

Darba izstrādes gaitā radās virkne ar secinājumiem, kas gan sakrīt ar teorijas daļā apskatītajām īpašībām, gan, manuprāt, nesakrīt.

Lai gan teorētiskais materiāls apgalvoja, ka pielāgot datu piekļuves objektu ir vieglāk nekā programmu, kas izmanto tiešo piekļuvi, taču izstrādājot programmas secināju, ka tas tomēr ir pietiekami piņķerīgi un laikietilpīgi. Tas tā liekas tieši tāpēc, ka ir vajadzīga spēja kārtīgi izdomāt visus iespējamos klupšanas akmeņus programmas izstrādē pirms kaut ko sākt darīt. Ne tikai pielāgošana var sagādāt problēmas, bet jau pirms programmas izveides var nākties ilgāk plānot izstrādes gaitu. No tā gan varu tikai izsecināt, ka tas ir ļoti lielā mērā atkarīgs no programmētāja, jo katram patīk citādāk programmēt.

Tā kā datu piekļuves objektu izmantošana prasa papildus klašu izveidi, programmas kods kļūst strukturētāks, līdz ar to arī vieglāk uztverams citiem, jo atbrīvo programmas kodu no liekas informācijas. Tātad labi izmantot, ja vairākiem programmētājiem jāstrādā pie viena projekta, jo tad lielāka iespēja, ka katra izmaiņas kritiski neietekmēs programmas kopējo darbību.

No tehniskās puses skatoties, datu piekļuves objekti izmanto daudz programmas atmiņas, bet tiešās piekļuves programmas biežāk sazinās ar datubāzi. No tā izriet, ja datubāze ir neliela un ar mazu veiktspēju, jāapsver iespēja izmantot datu piekļuves objektus. Ja klienta aparatūra ir salīdzinoši mazu veiktspēju, tad atkal iespējams labāk izmantot tiešo piekļuvi, jo samazināsies nepieciešamais atmiņas daudzums un daudzas lietas varēs iekļaut datubāzes vaicājumos, nevis ciklos.

Ja ir bailes uzsākt pašam veidot savus datu piekļuves objektus, var vispirms pavērot un pamācīties, kā tas jādara, izmantojot kādu datu piekļuves objektu ģeneratoru, kuri arī ir brīvi pieejami internetā.

IZMANTOTĀS LITERATŪRAS SARAKSTS

1. DAO Layer explained [Tiešsaiste] [Citēts 2013. gada 24. maijā] http://www.roseindia.net/struts/hibernate-spring/dao-layer-explained.shtml
2. Data Access Object [Tiešsaiste] [Citēts 2013. gada 24. maijā] http://www.codefutures.com/data-access-object/
3. Hibernate Generic Data Access Object [Tiešsaiste] [Citēts 2013. gada 25. maijā] http://spiproductsplatformsolution.wordpress.com/2012/10/30/hibernate-generic-data-access-object/
4. DaoGen Manula [Tiešsaiste] TitanicLinux.Net [Citēts 2013. gada 24. maijā] http://titaniclinux.net/cms/FC?link=daogen-manual
5. Core J2EE Patterns - Data Access Object [Tiešsaiste] [Citēts 2013.gadaa 28.maijā] http://www.oracle.com/technetwork/java/dataaccessobject-138824.html
6. Data Access Object Pattern [Tiešsaiste] Max Berger [Citēts 2013. gada 1. jūnijā] http://max.berger.name/research/silenus/print/dao.pdf
7. Integration using Direct Data Access [Tiešsaiste] Sasirekha [Citēts 2013. gada 1. jūnijā] http://itknowledgeexchange.techtarget.com/enterprise-IT-tech-trends/integration-using-direct-data-access/

PIELIKUMI

1. pielikums

Datu piekļuves objekta realizācijas pirmkods (StudentDaoImpl.java)

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao
{
   //visu studentu saraksts
   private List<Student> students = new ArrayList<Student>();
   private List<Course> courses = new ArrayList<Course>();

   //mainīgais, kas nodrošina savienojumu ar datubāzi
   private Connection con = null;

   private PreparedStatement deleteStudent = null;
   private PreparedStatement updateStudent = null;
   private PreparedStatement getAllStudents = null;
   private PreparedStatement addStudent = null;
   private PreparedStatement getLastID = null;

   //datubāzu vaicājumi
   private String getAllCoursesString = "SELECT Name, Teacher FROM kursi";
   private String getGradesString = "SELECT kursi.Name, kursi.Teacher,
atzimes.Grade " +
                                                    "FROM students,
atzimes, kursi " +
                                                    "WHERE students.ID
= ? AND students.ID = atzimes.Stud_ID AND kursi.ID = atzimes.Cours_ID ";
   private String deleteGradesString = "DELETE FROM atzimes WHERE Stud_ID =
?";
   private String deleteString = "DELETE FROM students WHERE Id = ? ";
   private String updateString = "UPDATE students SET Name = ?, Surname = ?,
Email = ? WHERE ID = ? ";
   private String getAllString = "SELECT * FROM students";
   private String addString = "INSERT INTO students (Name, Surname, Email)
VALUES (?,?,?)";
   private String getLastIDString = "SELECT LAST_INSERT_ID();";
   private String getCourseIDString = "SELECT ID FROM kursi WHERE Name = ?
AND Teacher = ?";
   private String addGradeString = "INSERT INTO atzimes VALUES(?,?,?)";
   private String getSingleGradeString = "SELECT * FROM atzimes WHERE Stud_ID
= ? AND Cours_ID = ?";
   private String updateGradesString = "UPDATE atzimes SET Grade = ? WHERE
STud_ID = ? AND Cours_ID = ?";

   //konstruktors
   public StudentDaoImpl()
   {
         //izveido savienojumu ar datubāzi
         try {
               String url = "jdbc:mysql://localhost:3306/";
               String dbName = "studenti";
               String driver = "com.mysql.jdbc.Driver";
               String userName = "test";
               String password = "test";
               Class.forName(driver).newInstance();
               con =
DriverManager.getConnection(url+dbName,userName,password);
               //papildus mainīgie SQL vaicājuma rezultātu glabāšanai
```

```
                ResultSet rs, rs2;
                //iegūst visus studentu datus no datubāzes un saglabā tos
sarkastā - students
                getAllStudents = con.prepareStatement(getAllString);
                rs = getAllStudents.executeQuery();

                Student st;
                List<Course> courseList;
                Course course;

                //cikls rezultātu kopas apstrādei
                while(rs.next())
                {
                        st = new Student();
                        st.setId(rs.getInt("ID"));
                        st.setName(rs.getString("Name"));
                        st.setSurname(rs.getString("Surname"));
                        st.setEmail(rs.getString("Email"));
                        //ar šo iegūstam katram studentam sarakstu ar viņa
kursiem
                        getAllStudents =
con.prepareStatement(getGradesString);
                        getAllStudents.setInt(1, st.getId());
                        rs2 = getAllStudents.executeQuery();
                        courseList = new ArrayList<Course>();
                        //cikls rezultātu kopas apstrādei
                        while(rs2.next())
                        {
                                course = new Course();
                                course.setName(rs2.getString("Name"));
                                course.setTeacher(rs2.getString("Teacher"));
                                course.setGrade(rs2.getInt("Grade"));
                                courseList.add(course);
                        }
                        st.setCourses(courseList);
                        students.add(st);
                }

                //iegūstam visus kursus un saglabājam arakstā - courses
                getAllStudents = con.prepareStatement(getAllCoursesString);
                rs = getAllStudents.executeQuery();
                while(rs.next())
                {
                        course = new Course(rs.getString("Name"),
rs.getString("Teacher3"));
                        courses.add(course);
                }

        } catch (Exception e) {
                e.printStackTrace();
        }
   }

   //studenta datu dzēšanas metode
   @Override
   public void deleteStudent(Student student)
   {
      try {
        //vispirms jāizdzēš datubāzes ieraksti par atzīmēm attiecīgajam
studentam
        deleteStudent = con.prepareStatement(deleteGradesString);
        deleteStudent.setInt(1, student.getId());
```

```
            deleteStudent.executeUpdate();
            //pēc tam var izdzēst paša studenta datus
            deleteStudent = con.prepareStatement(deleteString);
            deleteStudent.setInt(1,student.getId());
            deleteStudent.executeUpdate();

            //un protams arī jāizņem no studentu saraksta
            students.remove(student);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //iegūst sarakstu ar visiem studentiem un to atzīmēm
    @Override
    public List<Student> getAllStudents()
    {
        return students;
    }

    //atjauno informāciju datubāzē, ņemot vērā dotās izmaiņas
    @Override
    public void updateStudent(Student student)
    {
          try
          {
                //vispirsm pārbaudam vai nav mainījušies/pievienoti kādu
kursi  studentam
                List<Course> courses = student.getCourses();
                for(Course c: courses)
                {
                      //iegūstam kursa ID numuru
                      PreparedStatement ps =
con.prepareStatement(getCourseIDString);
                      ps.setString(1, c.getName());
                      ps.setString(2, c.getTeacher());
                      ResultSet rs = ps.executeQuery();
                      //rezultāts ir kursa ID
                      rs.next();
                      int courseID = rs.getInt("ID");

                      //iegūstam attiecīgā kursa atzīmi
                      ps = con.prepareStatement(getSingleGradeString);
                      ps.setInt(1, student.getId());
                      ps.setInt(2, courseID);
                      rs = ps.executeQuery();
                      //ja atzīme jau eksistē, tad veidojam UPDATE vaicājumu
                      if(rs.next())
                      {
                            ps = con.prepareStatement(updateGradesString);
                            ps.setInt(1, c.getGrade());
                            ps.setInt(2, student.getId());
                            ps.setInt(3, courseID);
                            ps.executeUpdate();
                      } else //ja neeksistē, tad veidojam jaunu ierakstu
datubāzē ar INSERT vaicājumu
                      {
                            ps = con.prepareStatement(addGradeString);
                            ps.setInt(1, student.getId());
                            ps.setInt(2, courseID);
                            ps.setInt(3, c.getGrade());
```

```
                        ps.executeUpdate();
                    }
                }
                //visbeidzot atjaunojtam studenta personas datus
                updateStudent = con.prepareStatement(updateString);
                updateStudent.setString(1, student.getName());
                updateStudent.setString(2, student.getSurname());
                updateStudent.setString(3, student.getEmail());
                updateStudent.setInt(4, student.getId());
                updateStudent.executeUpdate();
            }
            catch (SQLException e)
            {
                e.printStackTrace();
            }
    }

    //metode studenta izveidošanai datubāzē.
    @Override
    public void createStudent(Student student)
    {
            try
            {
                //izveidojam ierakstu ar studenta personas datiem, lai varētu
tālāk izmantot viņa ID numuru
                addStudent = con.prepareStatement(addString);
                addStudent.setString(1, student.getName());
                addStudent.setString(2, student.getSurname());
                addStudent.setString(3, student.getEmail());
                addStudent.executeUpdate();

                //iegūstam studentam piešķirto AUTO_INCREMENT vērtību
                getLastID = con.prepareStatement(getLastIDString);
                ResultSet rs = getLastID.executeQuery();
                rs.next();

                student.setId(rs.getInt(1));

                //cikls studenta kursu apstrādei
                for(Course c: student.getCourses())
                {
                    //iegūstam kursa ID numuru, lai varētu to sasaistīt
datubāzē
                    addStudent = con.prepareStatement(getCourseIDString);
                    addStudent.setString(1, c.getName());
                    addStudent.setString(2, c.getTeacher());
                    rs = addStudent.executeQuery();
                    rs.next();

                    int courseID = rs.getInt("ID");
                    //veicam ierakstu datubāzē par studenta sekmēm
                    addStudent = con.prepareStatement(addGradeString);
                    addStudent.setInt(1, student.getId());
                    addStudent.setInt(2, courseID);
                    addStudent.setInt(3, c.getGrade());
                    addStudent.executeUpdate();
                }
                //pievienojam studentu visu studentu sarakstam
                students.add(student);
            }
            catch (SQLException e)
            {
```

```
                e.printStackTrace();
        }
}

//atgriež visu kursu sarakstu
public List<Course> getAllCourses()
{
        return courses;
}
```

}

Datu piekļuves objekta testa programmas pirmkods (StudentDemo.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class StudentDemo
{
   public static void main(String[] args)
   {
         //datu piekļuves objekta izveide
      StudentDao studentDao = new StudentDaoImpl();
      //papildmainīgie dažādām programmas darbībām
      Student student;
      Course course;
      //mainīgais vērtību nolasīšanai no konsoles loga
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      int izvele = 0;
      try{

      //programmas galvenais cikls
      do
      {
       //darbību saraksta izvade uz ekrāna
         System.out.println("Izvelieties darbību:");
               System.out.println("1 - Apskatīt visus datus");
               System.out.println("2 - Rediģēt datus");
               System.out.println("3 - Ielikt/labot atzīmi");
               System.out.println("4 - Dzēst personas datus");
               System.out.println("5 - Apskatīt tikai viena studenta datus");
               System.out.println("6 - Pievienot jaunu studentu");
               System.out.println("7 - Beigt darbu");

               izvele = Integer.parseInt(br.readLine());

               System.out.println("----------------------------------------
");
               System.out.println("----------------------------------------
");
               //ja izvēle ir 1
               if(izvele == 1)
               {
                     //cikls visu studentu datu izvadei uz ekrāna
                     for (Student s : studentDao.getAllStudents())
                     {
                           System.out.println(s.getName() + " " +
s.getSurname());
                           //cikls studenta kursa datu izvadei uz ekrāna
                           for(Course c : s.getCourses())
                           {
                                 System.out.println(c.getName() + " - " +
c.getGrade());
                           }
                           System.out.println();
                     }

                     System.out.println();
                     System.out.println("Nospiediet Enter taustiņu lai tiktu
atpakaļ...");
                     br.readLine();
```

```
            } else if(izvele == 2) //ja izvēle ir 2
            {
                  //izvadam sarakstu ar studentiem un kārtās numuru, lai
vieglāk saprast ko izvēlēties
                  for (Student s : studentDao.getAllStudents())
                  {

      System.out.println((studentDao.getAllStudents().indexOf(s)+1) + ": " +
s.getName() + " " + s.getSurname());
                  }
                  System.out.println();

                  boolean izmainija = false;
                  //prasam, kura studenta datus rediģēt un to jaunās
vērtības
                  System.out.println("Ievadiet rediģējamā studenta kārtas
numuru.");
                  int index = Integer.parseInt(br.readLine());
                  System.out.println("Ievadiet vārdu, ja tas mainās: ");
                  String vards = br.readLine();
                  System.out.println("Ievadiet uzvārdu, ja tas mainās: ");
                  String uzvards = br.readLine();
                  System.out.println("Ievadiet e-pastu, ja tas mainās: ");
                  String epasts = br.readLine();

                  //iegūstam vēlamā studenta objektu
                  student = studentDao.getAllStudents().get(index - 1);

                  //pārbaudām vai vērtības ir ievadītas
                  if(vards.length() != 0)
                  {
                        student.setName(vards);
                        izmainija = true;
                  }
                  if(uzvards.length() != 0)
                  {
                        student.setSurname(uzvards);
                        izmainija = true;
                  }
                  if(epasts.length() != 0)
                  {
                        student.setEmail(epasts);
                        izmainija = true;
                  }

                  //vai ir veiktas izmaiņas
                  if(izmainija)
                  {
                        //ja ir tad atjaunojam datubāzi un studentu
sarakstu ar updateStudent() metodi
                        studentDao.updateStudent(student);

                        System.out.println();
                        System.out.println("Ieraksts veiksmīgi
atjaunots!");
                        System.out.println();
                  } else
                  {
                        System.out.println();
                        System.out.println("Personas dati netiks
mainīti!");
```

```
                        System.out.println();
                }

                System.out.println("Nospiediet Enter taustiņu lai tiktu
atpakaļ...");
                br.readLine();


            }else if(izvele == 3) //ja izvēle ir 3
            {
                //izvadam sarakstu ar studentiem un kārtās numuru, lai
vieglāk saprast ko izvēlēties
                for (Student s : studentDao.getAllStudents())
                {

      System.out.println((studentDao.getAllStudents().indexOf(s)+1) + ": " +
s.getName() + " " + s.getSurname());
                }
                System.out.println();

                System.out.println("Ievadiet rediģējamā studenta kārtas
numuru.");
                int studIndex = Integer.parseInt(br.readLine());
                System.out.println();

                //izvadam sarakstu ar kursiem, kurus apmeklē students,
lai vieglāk saprast ko izvēlēties
                for(Course c : studentDao.getAllCourses())
                {

      System.out.println((studentDao.getAllCourses().indexOf(c)+1) + ": " +
c.getName() + "(" + c.getTeacher() + ")");
                }
                System.out.println("Ievadiet kursa kārtas numuru.");
                int coursIndex = Integer.parseInt(br.readLine());
                //prasam atzīmi
                System.out.println("Ievadiet atzīmi: ");
                int atzime = Integer.parseInt(br.readLine());

                //iegūstam attiecīgos objektus
                student = studentDao.getAllStudents().get(studIndex - 1);
                course = studentDao.getAllCourses().get(coursIndex - 1);
                course.setGrade(atzime);

                //glabā vērtību vai jauns kurss vai jau esošs
                boolean jauns = true;
                //cikls tā noskaidrošanai
                for(Course c : student.getCourses())
                {
                        if(c.getName() == course.getName() &&
c.getTeacher() == c.getTeacher())
                        {
                                c.setGrade(atzime);
                                jauns = false;
                                break;
                        }
                }

                //ja jauns tad pievienojam to studenta sarakstam, ja nē,
tad tam atzīme jau ir atjaunota
                if(jauns)
                {
```

```
                        student.getCourses().add(course);
                    }
                    //atjaunojam datubāzi un sarakstu
                    studentDao.updateStudent(student);

                    System.out.println();
                    System.out.println("Nospiediet Enter taustiņu lai tiktu
atpakaļ...");
                    br.readLine();

            }else if(izvele == 4) //ja izvēle ir 4
            {
                    //izvadam sarakstu ar studentiem un kārtās numuru, lai
vieglāk saprast ko izvēlēties
                    for (Student s : studentDao.getAllStudents())
                    {

      System.out.println((studentDao.getAllStudents().indexOf(s)+1) + ": " +
s.getName() + " " + s.getSurname());
                    }
                    System.out.println();

                    System.out.println();
                    System.out.println("Ievadiet dzēšamā studenta kārtas
numuru.");
                    int index = Integer.parseInt(br.readLine());
                    //iegūstam attiecīgā studenta objektu
                    student = studentDao.getAllStudents().get(index - 1);
                    //izdzēšam studenta datus no datubāzes un saraksta
                    studentDao.deleteStudent(student);

                    System.out.println();
                    System.out.println("Ieraksts veiksmīgi izdzēsts!");
                    System.out.println();

                    System.out.println("Nospiediet Enter taustiņu lai tiktu
atpakaļ...");
                    br.readLine();

            }else if(izvele == 5) //ja izvēle ir 5
            {
                    //izvadam sarakstu ar studentiem un kārtās numuru, lai
vieglāk saprast ko izvēlēties
                    for (Student s : studentDao.getAllStudents())
                    {

      System.out.println((studentDao.getAllStudents().indexOf(s)+1) + ": " +
s.getName() + " " + s.getSurname());
                    }
                    System.out.println();

                    System.out.println("Ievadiet studenta kartas numuru.");
                    int index = Integer.parseInt(br.readLine());
                    //iegūstam studenta objektu
                    student = studentDao.getAllStudents().get(index - 1);

                    System.out.println(student.getName() + " " +
student.getSurname());
                    //cikls studenta kursu izvadei
                    for(Course c : student.getCourses())
                    {
```

```
                        System.out.println(c.getName() + " - " +
c.getGrade());
                    }
                    System.out.println();

                    System.out.println("Nospiediet Enter taustiņu lai tiktu
atpakaļ...");
                    br.readLine();
             }else if (izvele == 6) //ja izvēle ir 6
             {
                    //prasam ievadīt vajadzīgās vērtības
                    System.out.println("Ievadiet vārdu: ");
                    String vards = br.readLine();
                    System.out.println("Ievadiet uzvārdu: ");
                    String uzvards = br.readLine();
                    System.out.println("Ievadiet e-pastu: ");
                    String epasts = br.readLine();

                    student = new Student(vards, uzvards, epasts);

                    //izveidojam studenta objektu sarkastā un saglabājam tā
datus datubāzē.
                    studentDao.createStudent(student);

                    System.out.println();
                    System.out.println("Nospiediet Enter taustiņu lai tiktu
atpakaļ...");
                    br.readLine();

             } else //jebkurā citā gadījumā vienkārši beidzam darbu
             {
                    break;
             }

      } while (izvele != 7);
    } catch (Exception e)
    {
       e.printStackTrace();
    }

  }
}
```

3. pielikums

Datu tiešās piekļuves programmas kods (DirectAccess.java)

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;


public class DirectAccess
{
	//mainīgais programmas
	private static Connection con;

	//datubāzu vaicājumi, lai programma spētu pareizi funkcionēt
	private static String getAllCoursesString = "SELECT * FROM kursi";
	private static String isGradeCurrentString = "SELECT * FROM atzimes
WHERE Stud_ID = ? AND Cours_ID = ? ";
	private static String deleteGradesString = "DELETE FROM atzimes WHERE
Stud_ID = ?";
	private static String deleteStudentString = "DELETE FROM students
WHERE ID = ? ";
	private static String updateStudentString = "UPDATE students SET Name
= ?, Surname = ?, Email = ? WHERE ID = ? ";
	private static String getIdString = "SELECT * FROM students WHERE ID =
? ";
	private static String getAllString = "SELECT * FROM students";
	private static String addString = "INSERT INTO students (Name,
Surname, Email) VALUES (?,?,?)";
	private static String getStudentCoursesString = "SELECT Grade, Name
FROM atzimes, kursi WHERE ID = Cours_ID AND Stud_ID = ?";
	private static String updateGradeString = "UPDATE atzimes SET Grade =
? WHERE Stud_ID = ? AND Cours_ID = ?";
	private static String insertGradeString = "INSERT INTO atzimes
(Stud_ID, Cours_ID, Grade) VALUES (?,?,?)";

	public static void main(String[] args)
	{
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		int izvele = 0;
		//papildmainīgie darbam ar datubāzi
		ResultSet rs, rs2;
		PreparedStatement ps, ps2;
		try {
			//savienojuma parametri
			String url = "jdbc:mysql://localhost:3306/";
			String dbName = "studenti";
			String driver = "com.mysql.jdbc.Driver";
			String userName = "test";
			String password = "test";
			Class.forName(driver).newInstance();
			con =
DriverManager.getConnection(url+dbName,userName,password);
			//galvenais programmas cikls
			do
			{
				//izvēle
```

```
                        System.out.println("Izvelieties darbību:");
                        System.out.println("1 - Apskatīt visus datus");
                        System.out.println("2 - Rediģēt datus");
                        System.out.println("3 - Ielikt/labot atzīmi");
                        System.out.println("4 - Dzēst personas datus");
                        System.out.println("5 - Apskatīt tikai viena
studenta datus");
                        System.out.println("6 - Pievienot jaunu studentu");
                        System.out.println("7 - Beigt darbu");

                        izvele = Integer.parseInt(br.readLine());

                        System.out.println("------------------------------
------------");
                        System.out.println("------------------------------
------------");

                        //ja vēlas apsaktīt visus datus
                        if(izvele == 1)
                        {
                                //iegūst visu studentu datus
                                ps = con.prepareStatement(getAllString);
                                rs = ps.executeQuery();
                                //cikls rezultāta apstrādei
                                while(rs.next())
                                {
                                        //izvada studentu vārduus un uzvārdu
uz ekrāna

      System.out.println(rs.getString("Name") + " " +
      rs.getString("Surname"));
                                        //iegūst sattiecīgā studenta atzīmes
un kursus
                                        ps2 =
con.prepareStatement(getStudentCoursesString);
                                        ps2.setInt(1, rs.getInt("ID"));
                                        rs2 = ps2.executeQuery();
                                        while(rs2.next())
                                        {
                                                //izvada attiecīgā studenta
atzīmes un kursus

      System.out.println(rs2.getString("Name") + " - " +
rs2.getInt("Grade"));
                                        }
                                        System.out.println();
                                }

                                System.out.println();
                                System.out.println("Nospiediet Enter
taustiņu lai tiktu atpakaļ...");
                                br.readLine();

                        } else if(izvele == 2) //ja vēlas rediģēt datus
                        {
                                //izvada studentu datus, lai vieglāk
izvēlēties vajadzīgo
                                ps = con.prepareStatement(getAllString);
                                rs = ps.executeQuery();
                                while(rs.next())
                                {
```

```
                                        System.out.println(rs.getInt("ID") + "
- " +
                                        rs.getString("Name") + " " +
                                        rs.getString("Surname") + " - " +
                                        rs.getString("Email"));
                                }
                                //papildmainīgie
                                boolean irVards = true, irUzvards = true,
irEpasts = true;
                                //nepieciešamo parametru iegūšana
                                System.out.println();
                                System.out.println("Ievadiet nepieciešamā
studenta ID:");
                                int Id = Integer.parseInt(br.readLine());
                                System.out.println("Ievadiet vārdu, ja tas
mainās: ");
                                String vards = br.readLine();
                                System.out.println("Ievadiet uzvārdu, ja tas
mainās: ");
                                String uzvards = br.readLine();
                                System.out.println("Ievadiet epasta adresi,
ja tā mainās: ");
                                String epasts = br.readLine();
                                //pārbaude, vai visis parametri ievadīti
                                if(vards.length() == 0) irVards = false;
                                if(uzvards.length() == 0) irUzvards = false;
                                if(epasts.length() == 0) irEpasts = false;

                                if((!irVards || !irUzvards || !irEpasts) &&
!(irVards == irUzvards == irEpasts))
                                {
                                        //ja kāds nav ievadīts, tad jāiegūst
tā noklusējuma vērtība
                                        ps =
con.prepareStatement(getIdString);
                                        ps.setInt(1, Id);
                                        rs = ps.executeQuery();
                                        rs.next();
                                        if(!irVards) vards =
rs.getString("Name");
                                        if(!irUzvards) uzvards =
rs.getString("Surname");
                                        if(!irEpasts) epasts =
rs.getString("Email");
                                }
                                if(irVards || irUzvards || irEpasts)
                                {
                                        //ja visi parametri norādīti, tad
vienkārši atjaunina datus
                                        ps =
con.prepareStatement(updateStudentString);
                                        ps.setString(1, vards);
                                        ps.setString(2, uzvards);
                                        ps.setString(3, epasts);
                                        ps.setInt(4, Id);
                                        ps.executeUpdate();

                                        System.out.println();
                                        System.out.println("Ieraksts veiksmīgi
atjaunots!");
                                        System.out.println();
                                } else
```

```
                                {
                                        System.out.println("Izmaiņas netiks
veiktas!");
                                        System.out.println();
                                }

                                System.out.println("Nospiediet Enter
taustiņu lai tiktu atpakaļ...");
                                br.readLine();


                        }else if(izvele == 3) //ja vēlas piešķirt atzīmi
                        {
                                //attēlo studentu datus, lai ērtāk
izvēlēties
                                ps = con.prepareStatement(getAllString);
                                rs = ps.executeQuery();
                                while(rs.next())
                                {
                                        System.out.println(rs.getInt("ID") + "
- " +
                                        rs.getString("Name") + " " +
                                        rs.getString("Surname"));
                                }

                                //prasa izvēletā studenta ID numuru
                                System.out.println();
                                System.out.println("Ievadiet nepieciešamā
studenta ID:");
                                int studIndex =
Integer.parseInt(br.readLine());

                                //iegūst visus kursus
                                ps =
con.prepareStatement(getAllCoursesString);
                                rs = ps.executeQuery();
                                while(rs.next())
                                {
                                        System.out.println(rs.getInt("ID") + "
- " +
                                        rs.getString("Name") + " " +
                                        rs.getString("Teacher"));
                                }

                                //prasa ievadīt kursa ID numuru
                                System.out.println();
                                System.out.println("Ievadiet nepieciešamā
priekšmeta ID:");
                                int coursIndex =
Integer.parseInt(br.readLine());

                                //prasa ievadīt atzīmi
                                System.out.println("Ievadiet atzīmi:");
                                int atzime =
Integer.parseInt(br.readLine());
                                //pārbauda vai atzīme jau eksistē šādam
kursam un studentam
                                ps =
con.prepareStatement(isGradeCurrentString);
                                ps.setInt(1, studIndex);
                                ps.setInt(2, coursIndex);
                                rs = ps.executeQuery();
```

```
                                if(rs.next())
                                {
                                        //ja eskistē tad izmantojam UPDATE,
lai atjauninātu informāciju
                                        ps =
con.prepareStatement(updateGradeString);
                                        ps.setInt(1, atzime);
                                        ps.setInt(2, studIndex);
                                        ps.setInt(3, coursIndex);

                                        ps.executeUpdate();
                                } else
                                {
                                        //ja neeksistē, tad izveidojam jaunu
atzīmes ierakstu
                                        ps =
con.prepareStatement(insertGradeString);
                                        ps.setInt(1, studIndex);
                                        ps.setInt(2, coursIndex);
                                        ps.setInt(3, atzime);
                                        ps.executeUpdate();
                                }

                                System.out.println("Nospiediet Enter
taustiņu lai tiktu atpakaļ...");
                                br.readLine();

                        }else if(izvele == 4) //ja vēlas dzēst studenta
datus
                        {
                                //studentu saraksta attēlošana
                                ps = con.prepareStatement(getAllString);
                                rs = ps.executeQuery();
                                while(rs.next())
                                {
                                        System.out.println(rs.getInt("ID") + "
- " +
                                        rs.getString("Name") + " " +
                                        rs.getString("Surname"));
                                }
                                //prasa studenta ID numuru
                                System.out.println();
                                System.out.println("Ievadiet nepieciešamā
studenta ID:");
                                int Id = Integer.parseInt(br.readLine());
                                //vispirms jāizdzēš visas studemta atzīmes
no datubāzes
                                ps =
con.prepareStatement(deleteGradesString);
                                ps.setInt(1, Id);
                                ps.executeUpdate();
                                //pēc tam var izdzēst arī datus par pašu
studentu
                                ps =
con.prepareStatement(deleteStudentString);
                                ps.setInt(1, Id);
                                ps.executeUpdate();

                                System.out.println();
                                System.out.println("Ieraksts veiksmīgi
izdzēsts!");
```

```
                                System.out.println();

                                System.out.println("Nospiediet Enter
taustiņu lai tiktu atpakaļ...");
                                br.readLine();

                        }else if(izvele == 5) //ja vēlas apskatīti tikai
viena studenta datus
                        {
                                //studentu saraksts izvēlei
                                ps = con.prepareStatement(getAllString);
                                rs = ps.executeQuery();
                                while(rs.next())
                                {
                                        System.out.println(rs.getInt("ID") + "
- " +
                                        rs.getString("Name") + " " +
                                        rs.getString("Surname"));
                                }
                                //prasa studenta ID numuru
                                System.out.println();
                                System.out.println("Ievadiet nepieciešamā
studenta ID:");
                                int Id = Integer.parseInt(br.readLine());
                                //Izvadam studenta vārdu un uzvārdu
                                ps = con.prepareStatement(getIdString);
                                ps.setInt(1, Id);
                                rs = ps.executeQuery();
                                rs.next();

                                System.out.println(rs.getString("Name") + "
" + rs.getString("Surname") + " - " + rs.getInt("Score"));
                                //un zem tā kursus un atzīmes
                                ps =
con.prepareStatement(getStudentCoursesString);
                                ps.setInt(1, Id);
                                rs = ps.executeQuery();
                                //cikls, lai apstrādātu visus kursus
                                while(rs.next())
                                {

        System.out.println(rs.getString("Name") + " - " + rs.getInt("Grade"));
                                }
                                System.out.println();

                                System.out.println("Nospiediet Enter
taustiņu lai tiktu atpakaļ...");
                                br.readLine();

                        } else if(izvele == 6) //ja vēlas izveidot ierakstu
par studentu
                        {
                                //prasa ievadīt parametrus
                                System.out.println();
                                System.out.println("Ievadiet vārdu: ");
                                String vards = br.readLine();
                                System.out.println("Ievadiet uzvārdu: ");
                                String uzvards = br.readLine();
                                System.out.println("Ievadiet epastu: ");
                                String epasts = br.readLine();
                                //saglabā datus datubāzē
                                ps = con.prepareStatement(addString);
```

```
                                    ps.setString(1, vards);
                                    ps.setString(2, uzvards);
                                    ps.setString(3, epasts);
                                    ps.executeUpdate();

                                    System.out.println("Nospiediet Enter
taustiņu lai tiktu atpakaļ...");
                                    br.readLine();
                              } else
                              {
                                    break;
                              }
                        } while (izvele != 7);
                  } catch (Exception e)
                  {
                        e.printStackTrace();
                  }
            }

      }
```

DOKUMENTĀRĀ LAPA

Bakalaura darbs „Datu piekļuves objekti SQL datubāzēm” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Andris Balodis

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: Dr.dat. Jānis Zuters

Recenzents: Dr.dat. Uldis Straujums

Darbs iesniegts Datorikas fakultātē 04.06.2013.

Dekāna pilnvarotā persona: metodiķe Ārija Sproģe

Darbs aizstāvēts bakalaura pārbaudījumus komisijas sēdē

_____prot. Nr. ___, vērtējums _____

Komisijas sekretāre: