RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Lietišķo datorsistēmu institūts

Studiju darbs mācību priekšmetā

„Algoritmi un programmēšanas metodes”

Transakciju analīze

Izstrādāja: Sergejs Terentjevs

3. kurss, 4.grupa

Apl.nr. 061RDB140

Pārbaudīja: prof. S. Kozlova

\

2009./2010. māc. g.

Saturs

[1. Transakcijas jēdziens 3](#_Toc263305025)

[1.1. Transakcijas īpašības 3](#_Toc263305026)

[1.2. Transakciju viedi 4](#_Toc263305027)

[1.3. Relāciju datu bāžu transakciju operatori 5](#_Toc263305028)

[2. Programmatūras piemērs 7](#_Toc263305029)

[Izmantota literatūra 11](#_Toc263305030)

# Transakcijas jēdziens

Transakcija ir viena fundamentālam relāciju datu bāžu koncepcijām, kuras pamatā ir mehānisms veicamo izmaiņu vadībai. Transakcija paredz secīgo, nedalāmo operāciju izpildes vadību, kurām izpildoties veicamas izmaiņas tiek apstiprinātas un pretēji, jā kāda no operācijām neizpildās, visas izmaiņas tiek atceltas.

Transakcijas nodrošina datu integritāti un ļauj izvairīties no nevēlamam situācijām, kuras var rasties kļūdaini izpildoties operāciju virknei. Šādas situācijas piemērs ir naudas pārskatījuma operācijas veikšana. Piemēram, jā nepieciešams veikt kādas noteiktas naudas summas pārskatījumu no konta A uz kontu B. Šāda operācija sastāv no diviem pamatoperācijām:

1. Noteiktas naudas summas izņemšana no konta A,
2. Dotas summas pārskaitīšana uz kontu B.

Šajā gadījumā, jā pirmā operācija tiks veiksmīgi izpildīta un otras operācijas izpildes rezultātā radusies kļūda, tad kopējs operāciju izpildes rezultāts būs kļūdains, ka rezultāta tiks nozaudēta no konta A izņemta naudas summa. Transakcijas ļauj izvairīties no šādam situācijām, apstiprinot veiktas izmaiņas tikai tad, kad ir veiksmīgi izpildījusies otra operācija.

## Transakcijas īpašības

Transakcijām ir raksturīgas četras pamatīpašības, kuras tiek apzīmētas ar terminu ACID. Šādas īpašības ir:

1. Atomitāte (*atomicity*) – raksturo transakcijas izpildes nosacījumu. Saskaņa ar šo nosacījumu, transakcija izpildās tikai tad, kad secīgi izpildās visas tajā iekļautas operācijas, citādi transakcija neizpildās.
2. Saskaņotība (*consistency*) – raksturo transakcijas ietekmi uz datu bāzes stāvokli. Saskaņa ar doto īpašību, pirms un pēc transakcijas izpildes datu bāzes struktūrai ir jābūt stabilai (tabulām un to attiecībām jābūt neskartām).
3. Nošķirtība (*isolation*) – raksturo transakciju savstarpējas ietekmes pakāpi. Katra transakcija ir nošķirta, tās nevar ietekmēt uz citam paralēli pildāmo transakcijām. Tādējādi viena transakcija nevar darboties ar citas transakcijas starp līmeni, kamēr tā nav izpildījusies.
4. Ilgderīgums (*durability*) – pēc transakcijas izpildes, tai ir jāparedz veikto izmaiņu saglabāšanu datu bāzē.

## Transakciju viedi

Transakcijas iedalās vairākas kategorijas:

* Klasiskas – vienkāršas transakcijas, kuras balstās uz ACID īpašībām. Tās paredz vienotas operāciju virknes izpildes kontroli, kuras izpildes rezultātā izmaiņas vai nu tiek saglabātas vai arī atceltas.
* Cikliskās – paredz izpildes operāciju dalīšanu. Transakciju pamāta ir mehānisms, kurš piedāvā kāda noteikta transakcijas izpildes posma saglabāšanu. Izmantojot šādu iespēju ir iespējams atgriezties uz saglabāto transakcijas izpildes posmu, gadījumā jā transakcijas turpmākas izpildes gaitā ir radusies kļūda. Cikliskas transakcijas izpildes algoritma piemērs ir attēlots attēlā 1.1.



* 1. att. Cikliskas transakcijas izpildes algoritma blokshēma.
* Iegultas – mehānisma pamata ir viena galvenā aktīva transakcija, kura pilda citu apakš (klasisko) transakciju vadību. Šāda transakciju hierarhija tiek izveidota neaizvērot galveno transakciju, kuras ietvaros tiek veidota jauna transakcija. Transakciju kopējais izpildes rezultāts ir atkarīgs no katras apakš transakcijas izpildes rezultāta, t.i., jā apakš transakcija tiks atcelta vai arī tās izpildes rezultāta radīsies kļūdas, tad visas transakcijas (ieskaitot galveno) tiks atceltas. Transakciju izpildes rezultāta veiktas izmaiņas tiek fiksētas tikai tad, kad visas transakcijas ir izpildījušas veiksmīgi.

## Relāciju datu bāžu transakciju operatori

Transakciju vadība tiek realizēta ar speciālo operatoru palīdzību. Dotie operatori tiek pielietoti transakciju kontekstā, nosakot izmaiņu fiksēšanas, atcelšanas u.tml. nepieciešamību. Šādi operatori ir:

* *BEGIN* – operators pilda transakcijas inicializēšanu. Dažādas relāciju datu bāzēs dota operatora izsaukšanas komanda varbūt atšķirīga, piemēram, SQL Server gadījuma šāda komanda ir *BEGIN TRANSACTION*, savukārt, Oracle datu bāzes gadījumā šāda komanda ir *START TRANSACTION* u.t.t. Papildus vairākas datu bāzes šāda operatora tieša noradīšana nav nepieciešama, transakcija automātiski tiks inicializēta vadoties pēc transakcijas kontekstā turpmāk lietotiem transakciju operatoriem.
* *COMMIT* – operators raksturo transakcijas ietvaros veikto izmaiņu fiksēšanas nepieciešamību. Dažas relāciju datu bāzēs, piemēram, Oracle, dotam operatoram varbūt noteikti papildus parametri, kuri nosaka jaunas transakcijas izveidošanas nepieciešamību pēc dotas transakcijas fiksēšanas.
* *ROLLBACK* – operators raksturo transakcijas ietvaros veikto izmaiņu atcelšanas nepieciešamību. Dotam transakcijas operatoram var tikt uzdots parametrs, kurš atcels visas notikušas izmaiņas līdz kādam noteiktām transakciju saglabātam izpildes posmam (kontroles punktam).
* *SAVEPOINT* – operators ļauj uzstādīt kontroles punktu, t.i., saglabāt transakcijas pašreizējo izpildes posma veiktas izmaiņas, pie kuram ir iespējams atgriezts turpmākas transakcijas izpildes gaitā. Doto operatora nosaukums dažādas relāciju datu bāzes varbūt atšķirīgs, piemēram, SQL Server gadījuma tās tiek izsaukts ar *SAVE* komandas palīdzību. Dotie kontroles punkti ir ciklisko transakciju neatņemama sastāvdaļa.

Sākotnēji standartā ANSI/SQL bija noteikti tikai divi transakciju operatori: *COMMIT* un *ROLLBACK*. Turpmāk transakciju modelis tika paplašināts, ieviešot *BEGIN* un *SAVEPOINT* operatorus, kas pirmo tika izdarīts kompānijas SYBASE komerciālajā datu bāzē.

Doto operatoru pielietošanas piemēri ir attēloti attēlos 1.2 un 1.3. Attēlā 1.2 ir paradīts SQL Server datu bāzes iegultas transakcijas piemērs, kura atsevišķos transakcijas blokos dažādas tabulas tiek iestarpināti jauni ieraksti. Pēc attēla ir redzams, ka katra atsevišķa transakcija tiek inicializēta ar *BEGIN* operatora palīdzību un visu transakciju izpildes rezultātā veiktas izmaiņas tiek fiksētas ar *COMMIT* operatora palīdzību. Dota transakciju kopējs rezultāts tiks fiksēts tikai tajā gadījumā, ja visas transakciju operācijas būs veiksmīgi izpildītas.

|  |  |
| --- | --- |
| 01 | BEGIN TRANSACTION |
| 02 | INSERT INTO Grāmata (GrāmatasID, Nosaukums, Izdevniecība) |
| 03 | VALUES (6, 'Datu bāzes', 'Zvaigzne ABC'); |
| 04 | BEGIN TRANSACTION |
| 05 | INSERT INTO Autors (AutoraID, Vārds, Uzvārds) |
| 06 | VALUES (5, 'Aleksejs', 'Vasiljevs'); |
| 07 | BEGIN TRANSACTION |
| 08 | INSERT INTO Grāmata\_Autors (GrāmatasID, AutoraID) |
| 09 | VALUES (6, 5); |
| 10 | COMMIT |

* 1. att. Iegultas transakcijas piemērs.

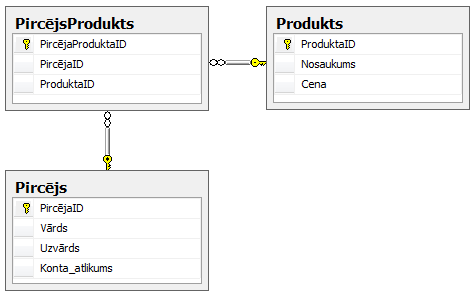
Attēlā 1.3 ir paradīts cikliskas transakcijas piemērs, kurā pēc katras operācijas izpildes tiek īslaicīgi saglabātas veiktas izmaiņas (kontroles punkti). Attēla priekšpēdējā rindkopā tiek izsaukts *ROLLBACK* operators, kurš atgriež pašreizējo datu bāzes stāvokli pirms otras atjaunošanas (*update*) operācijas izpildes, kurš arī tiek fiksēts ar operatora *COMMIT* palīdzību.

|  |  |
| --- | --- |
| 01 | BEGIN TRANSACTION |
| 02 | UPDATE Grāmata SET Nosaukums = 'DBVS' WHERE GrāmatasID = 6; |
| 03 | SAVE TRANSACTION KontrolesPunktsA; |
| 04 | UPDATE Autors SET Vārds = 'Mihails' WHERE AutoraID = 5; |
| 05 | SAVE TRANSACTION KontrolesPunktsB; |
| 06 | ROLLBACK TRANSACTION KontrolesPunktsA; |
| 07 | COMMIT |

* 1. att. Cikliskas transakcijas piemērs.

# Programmatūras piemērs

Transakcijas pielietošanas piemēra demonstrēšanai tika izstrādāta preču pasūtīšanas sistēma. Dotas sistēmas mērķis ir piedāvāt lietotājiem preču pasūtīšanas iespējas. Dota mērķa īstenošanai tika izstrādāta attēla 2.1 redzama datu bāzes struktūra.



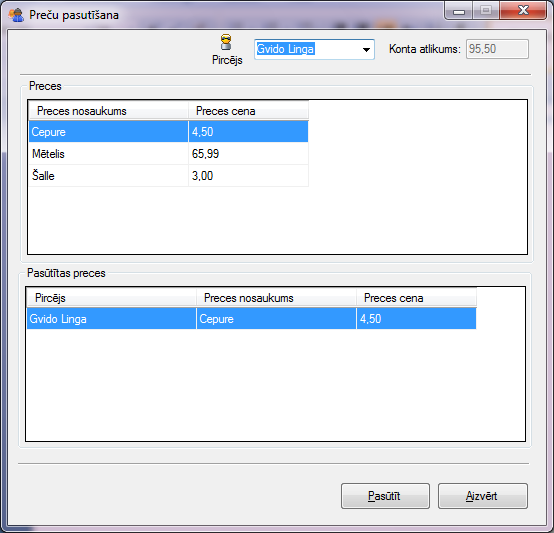
* 1. att. Preču pasūtīšanas sistēmas datu bāzes struktūra.

Tabula „Pircējs” satur informāciju par lietotājiem, kuri veic attiecīgo preču pasūtīšanu. Tā glabā datus par pircēju vārdiem, uzvārdiem un viņu konta pieejamo naudas summu. Tabula „Produkts” satur informāciju par preces nosaukums un cenu. Par cik starp abām tabulām eksistē „daudzi pret daudziem” attiecība (vienai precei var būt vairāki pircēji un vienām pircējam var būt vairākas preces), datu bāzes struktūra ir paredzēta „PircējsProdukts” tabula. Dota tabula veic pircēju un preču (produktu) tabulu saistīšana, glabājot informāciju par lietotāju veiktajiem pasūtījumiem. Šādus pasūtījumus sastāda pasūtītāja un preces unikālie identifikatori.

Programmas preču pasūtīšanas forma ir attēlota 2.2 attēlā. Dota forma izvada informāciju par pircēju, t.i., viņa vārdu, uzvārdu, konta naudas summas atlikumu un pasūtīto preču sarakstu. Papildus tā arī izvada sarakstu par datu bāzē pieejamam precēm, kuras pircējs var iegadāties. Preču pasūtīšanai, lietotājam ir nepieciešams izvelēties vēlamo preci no saraksta un nospiest pogu „Pasūtīt”.

Preču pasūtīšanas operācija sastāv no divām pamatoperācijām:

* Saglabāt pircēja pasūtījumu „PircējsProdukts” tabulā.
* Atņemt no lietotāja konta pasūtītas preces cenas vērtību. Dota darbība paredz tabulas „Pircējs” konta atlikuma lauka atjaunošanu.

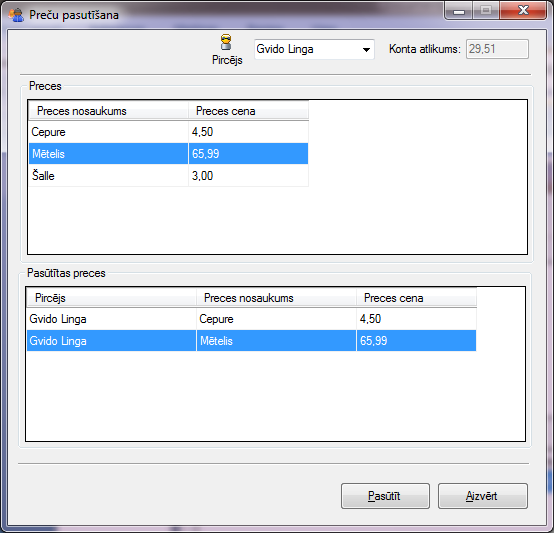


* 1. att. Preču pasūtīšanas forma.

Preces pasūtīšanas izpildes rezultāts ir attēlots attēlā 2.3. Tajā ir imitēta lietotāja Gvido Linga mēteļa preces pasūtīšana, kuras rezultātā no lietotāja konta atlikuma ir atņemta atbilstoša naudas summa (preces cena) un pasūtīto preču saraksta ir parādījusies jauna pasūtīta prece.

Programmas pirmtekstā šādas pamatoperācijas ir iekļautas transakcijas kontekstā, kura garantē izmaiņu fiksēšanu tikai tad, kad ir korekti izpildījušas abas pamatoperācijas. Transakcijas pirmteksta fragments ir attēlots attēla 2.4.

Programmas izstrāde ir veikta C# programmēšanas valodā, pielietojot ADO.NET tehnoloģiju. Dota tehnoloģija piedāvā vairākus datu gādniekus ar kuru palīdzību tiek nodrošināta pieeja dažādiem datu avotiem. Katrs tehnoloģijas datu gādnieks paredz vairākus objektus, ar kuru palīdzību tiek veiktas nepieciešamas operācijas ar datu bāzi.



* 1. att. Preces pasūtīšanas izpildes rezultāts.

Dota piemēra gadījuma šādi objekti ir:

* *SqlConnection* – pilda savienojuma ar SQL Server datu bāzi vadību. Objekts satur savienojuma atvēršanas, aizvēršanas, komandas objekta piesaistīšanas, transakciju inicializēšanas u.c. metodes.
* *SqlCommand* – nodrošina SQL vaicājumu izpildīšanu un iegūtas ierakstu kopas atgriešanu, piedāvā metodes parametru (datu bāžu mainīgo) uzdošanai u.tml.
* *SqlTransaction ­*– objekts pilda inicializētas SQL Server transakcijas vadību.

|  |  |
| --- | --- |
| 01 | // Savienojuma objekta izveidošana |
| 02 | using (SqlConnection sqlConn = new SqlConnection(connectionString)) { |
| 03 | // Komandas objekta izveidošana |
| 04 | SqlCommand sqlComm = sqlConn.CreateCommand(); |
| 05 | SqlTransaction sqlTran = null; |
| 06 |  |
| 07 | try { |
| 08 | // Savienojuma atvēršana |
| 09 | sqlConn.Open(); |
| 10 | /\* Parametru uzdošana (tiek izveidoti mainīgie pircēja, preces |
| 11 | identifikatoru un konta atlikušas naudas summas glabāšanai) \*/ |
| 12 | sqlComm.Parameters.AddWithValue("@PircējaID", pircejaID); |
| 13 | sqlComm.Parameters.AddWithValue("@ProduktaID",(int)gridProduct |
| 14 | .SelectedRows[0].Cells["ProduktaID"].Value); |
| 15 | sqlComm.Parameters.AddWithValue("@Atlikums", aprekinAtlikums); |
| 16 | // Transakcijas inicializācija |
| 17 | sqlTran = sqlConn.BeginTransaction(); |
| 18 | // Transakcijas saistīšana ar objekta pildāmam komandām |
| 19 | sqlComm.Transaction = sqlTran; |
| 20 |  |
| 21 | // SQL vaicājumu izpilde |
| 22 | string sql = @"INSERT INTO PircējsProdukts (PircējaID, ProduktaID) |
| 23 | VALUES (@PircējaID, @ProduktaID);"; |
| 24 | sqlComm.CommandText = sql; |
| 25 | sqlComm.ExecuteNonQuery(); |
| 26 |  |
| 27 | sql = @"UPDATE Pircējs SET Konta\_atlikums = @Atlikums |
| 28 | WHERE PircējaID = @PircējaID;"; |
| 29 | sqlComm.CommandText = sql; |
| 30 | sqlComm.ExecuteNonQuery(); |
| 31 |  |
| 32 | // Transakcijas fiksēšana |
| 33 | sqlTran.Commit(); |
| 34 | } catch (SqlException ex) { |
| 35 | // Izņēmuma gadījumā paredzēt izmaiņu atcelšana |
| 36 | sqlTran.Rollback(); |
| 37 | throw ex; |
| 38 | } finally { |
| 39 | // Savienojuma aizvēršana |
| 40 | sqlConn.Close(); |
| 41 | } |
| 42 | } |

* 1. att. Pasūtījuma veikšanas operācijas pirmteksts.

# Izmantota literatūra

1. Крёнке Д. Теория и практика построения баз данных 8-е издание. – Москва: Питер, 2003. – 799 lpp.
2. Карпова С. Т. Модели транзакций / Internets. –

<http://www.intuit.ru/department/database/dbmdi/11/>.

1. Полякова Н. Л. Основы SQL / Internets. –

<http://www.intuit.ru/department/database/sql/16/1.html>.

1. C# 2008 и платформа .NET 3.5 для профессионалов / К. Нейгел, Б. Ивьен, Д. Глинн и др. – Москва: «Диалектика», 2009. – 1738 lpp.

1. Малик С. Microsoft ADO.NET 2.0 для профессионалов – Москва: Издательский дом «Вильямс», 2006. – 553 lpp.