Rīgas Tehniska universitāte

Datorzinātnes un Informācijas tehnoloģijas fakultāte

Trešais praktiskais darbs mācību priekšmetā

„Lielas datu bāzes”

Servera programmēšana (PL/SQL)

Izstrādāja: Sergejs Terentjevs

3. kurss, 4.grupa

Apl.nr. 061RDB140

Pārbaudīja: Prof. J. Eiduks

2007./2008. māc. g.

# ANOTĀCIJA

Praktiska darbā tika izveidota un izvadīta datu bāzes datu glabāšanas struktūra, izveidots PL/SQL anonīmo bloku, izstrādātas procedūras, praktiski pielietots kursors, izstrādāta programmu pakotne, funkcija, kura atgriež tabulveida struktūru, ORDER, MEMEBER, MAP tipa metodes, izstrādāts kursors un izveidota 1:N objektu saite.

Praktiskais darbs tika izpildīts Oracle 10g Express Edition datu bāzes vadības sistēmā uz Genuine Intel tipā procesorā Windows XP operētājsistēmā.

Laboratorijas darba pārskata apjoms: 40 lappuses izklāsta tekstā, tajā ietilpst 13 nodaļas, 2 shēmas, 8 attēli un literatūras sarakstā ir norādīti 3 literatūras avoti.

# SATURS

[ANOTĀCIJA 2](#_Toc217365317)

[SATURS 3](#_Toc217365318)

[UZDEVUMA NOSTĀDNE 4](#_Toc217365319)

[DATU BĀZES STRUKTŪRAS PAMATOJUMS 5](#_Toc217365320)

[DATU BĀZES DATU STRUKTŪRAS IZVEIDOŠANA 7](#_Toc217365321)

[DATU IEVADE OBJEKTU TABULAS 11](#_Toc217365322)

[DATU BĀZES DATU STRUKTŪRAS IZVADĪŠANA 13](#_Toc217365323)

[ANONĪMA PL/SQL BLOKA IZVEIDOŠANA 16](#_Toc217365324)

[PROCEDŪRAS IZVEIDOŠANA 18](#_Toc217365325)

[PROCEDŪRAS IZVEIDOŠANA, KURA OPERĒ AR KURSORU 20](#_Toc217365326)

[FUNKCIJU IZVEIDOŠANA 24](#_Toc217365327)

[ORDER TIPA METODES IZVEIDOŠANA OBJEKTU SKATAM 27](#_Toc217365328)

[MEMBER TIPA METODES IZVEIDOŠANA OBJEKTU SKATAM 29](#_Toc217365329)

[MAP TIPA METODES IZVEIDOŠANA OBJEKTU SKATAM 31](#_Toc217365330)

[TRIGERA IZSTRADĀŠANA 32](#_Toc217365331)

[PROCEDŪRAS IZVEIDOŠANA TABULĀM AR REALIZĒTU 1:N OBJEKTU SAITI 34](#_Toc217365332)

[SECINĀJUMI 39](#_Toc217365333)

[IZMANTOTA LITERATŪRA 40](#_Toc217365334)

# UZDEVUMA NOSTĀDNE

1. Izveidot un izvadīt datu bāzes datu glabāšanas struktūru,
2. Izveidot PL/SQL anonīmo bloku,
3. Izstrādāt procedūru,
4. Izstrādāt procedūru, kurā operē ar kursoru,
5. Izveidot funkciju:
6. Funkcijas atgriežama vērtība ir tabulveida struktūra,
7. Funkcija tiek definēta programmu PL/SQL pakotnē.
8. Izveidot Order tipa metodi un vaicājumu ar šo metodi objektu skatam,
9. Izveidot Member tipa metodi un vaicājumu ar šo metodi objektu skatam,
10. Izveidot Map tipa metodi un vaicājumu ar šo metodi objektu skatam,
11. Izstrādāt trigeri,
12. Izveidot procedūru tabulām ar 1:N objekti saiti,
13. Dod secinājumus veiktajām darbam.

# DATU BĀZES STRUKTŪRAS PAMATOJUMS

Praktiskajā uzdevumā tiek risināta preču pasūtīšanas problēma kompānijā, risinot šo problēmu tiks izstrādāta neliela datu bāze, kurās pamatā būs četri objektu tipi: Darbinieks, Pasūtījums, Prece un Piegādātāji.

Objektu tips Darbinieks saturēs tādus darbinieku raksturojušus atribūtus kā unikāls identifikators, vārds, uzvārds, telefona numurs, alga, prēmija un amats. Visus darbiniekus mēs varam sakārtot pēc kāda raksturojoša raksturlieluma, šīm nolūkam objektu tipā Darbinieki tiks izveidota Map tipa metode Samaksa, kurā ļaus mums sakārtot darbinieku eksemplārus objektu tabulā vai arī objektu skatā pēc to ienākuma lieluma, kuru sastāda algas un prēmijas kopsummā.

Cits objektu tips Prece saturēs tādus atribūtus kā preces unikāls identifikators, nosaukums, cena, PVN, atlaide. Bieži vien kompānijai interesē preces galēja cena, kuru sastāda preces cena kopā ar PVN un no šī rezultāta atņemot atlaides vērtību, šīm nolūkam objektu tipā tiks izveidots vēl viens atribūts Maksa, kurš arī saturēs šīs preces galējo vērtību. Lai lietotājiem nenāktos pašiem skaitīt preces galējo vērtību, tiks izveidota Member tipa metode Gala\_cena, kura arī aprēķinās kāda objekta eksemplāra galējo cenu.

Objektu tips Pasūtījumi saturēs tādus raksturojušus atribūtus kā unikāls identifikators, pasūtītas preces un pasūtījumu veikuša darbinieka identifikatorus, pasūtījuma veikšanas datumu, pasūtītas preces skaitu un atribūtu status, kurš norāda vai dota prece ir piegādāta (1) vai arī nav (0). Tips saturēs arī Order tipa metodi Kartot\_Datum, kurā ļaus sakārtot doto objektu eksemplārus objekta tabulā vai arī objektu skatā pēc pasūtīšanas datuma.

Objektu tips piegādātāji saturēs atribūtus unikāls identifikators, piegādātāju kompānijas nosaukums.

Uz šo objektu tipu pamatā tiks izveidotās četras objektu tabulas: Darbinieki (objektu tips Darbinieks), Piegādātāji (objektu tips Piegādātājs), Preces (objektu tips Prece), Pasūtījumi (objektu tips Pasūtījums). Tātad objektu tabulā Darbinieki tiks glabāta informācija par darbiniekiem, objektu tabula Pasūtījumi glabās darbinieku veiktos pasūtījumus, objektu tabula Preces glabās informāciju par pieejamām precēm, kurās var pasūtīt un objektu tabula Piegādātāji saturēs informāciju par piegādātājiem, kuri piegadās pasūtīto preci kompānijai. Objektu tabulas ir savstarpēji saistītas, to saites un objektu tabulu struktūras ir paradītas 1. shēmā.

**Piegadataji**

PiegID

Nosaukums

**Preces**

PreceID

Nosaukums

Cena

PVN

Atlaide

Maksa

Gala\_Cena()

PiegID

**Darbinieki**

DarbID

Vards

Uzvards

Amats

Tel\_Num

Alga

Premija

Samaksa()

**Pasutijumi**

PasutID

DarbID

PreceID

PasutDatums

Skaits

Status

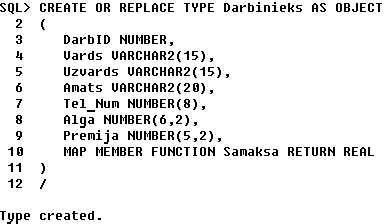
Kartot-Datum(p Pasutijums)

1. shēma – datu bāzes struktūras shēma.

# DATU BĀZES DATU STRUKTŪRAS IZVEIDOŠANA

Šajā nodaļā izveidosim iepriekšēja nodaļā minētos objektu tipus un to objektu tabulas, lai turpmāk mēs varētu veiksmīgi konstruēt vairākus datu manipulācijas un izgūšanas vaicājumus, ka arī izveidot vairākas funkcijas un procedūras, kurās operē objektu tabulas datiem.

Vispirms izveidosim objekta tipu Darbinieks, kurš saturēs visus iepriekšēja nodaļā apskatītos atribūtus un Map tipa metodi Samaksa, kurā izpildes rezultātā atgriezis kādu reālo skaitli. Šīm nolūkam, sastādām šādu pierakstu:



Objekta tipa shematiskais attēls ir paradīts attēlā 1. Tajā paradīts objekts ar 7 atribūtiem.

1. attēls – Objekta tipa Darbinieks shematisks attēls.

Pēc objekta tipa izveidošanās varam izveidot objektu tabulu Darbinieki, kurā glābās mūsu izveidota objekta tipa eksemplārus un to atribūtu vērtības, šīm nolūkam veicām šādu pierakstu:

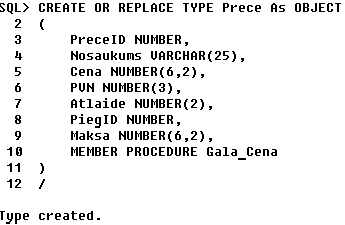


Objektu tabulas shematiskais attēls ir paradīts attēlā 2.

.............

1. attēls – Objekta tabulas Darbinieki shematisks attēls.

Līdzīgi izveidosim objekta tipu Prece. Šīm nolūkam, sastādām šādu pierakstu:



Objekta tipa un to atribūtu shematiskai attēls ir paradīts attēlā 3.

1. attēls – Objekta tipa Prece shematisks attēls.

Pēc objekta tipa izveidošanas veidojām objekta tabulu Preces, objektu tabulas veidošanas pieraksts ir šāds:

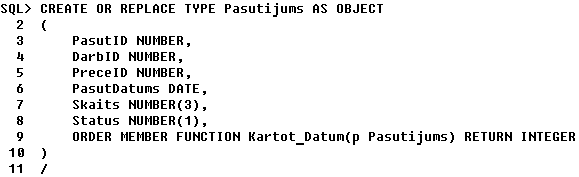


Pats objektu tabulas shematisks attēls ir parādīts attēlā 4.

.............

1. attēls – Objekta tabulas Preces shematisks attēls.

Kā nākamo izveidosim objektu tipu Pasūtījums, šeit ir jāpiebilst kā iepriekšēja nodaļā apskatīta metode Kartot\_Datum izpildes rezultātā atgriež kādu veselu skaitli, kurš attiecīgi noteiks objekta eksemplāra pozīciju sakārtojumā. Šīm nolūkam veicām šādu pierakstu:



Objekta tipa shematiskai attēls ir paradīts attēlā 5. Tajā paradīts objekts ar 6 atribūtiem.

1. attēls – Objekta tipa Pasūtījums shematisks attēls.

Līdzīgi kā iepriekšējos gadījumos arī šīm objekta tipam izveidosim objektu tabulu. Pasūtījuma objektu tabulas veidošanas pieraksts ir sekojošs:

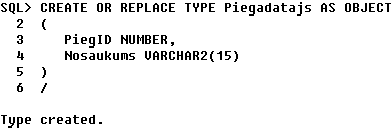


Pats objektu tabulas shematisks attēls ir parādīts attēlā 6.

.............

1. attēls – Objekta tabulas Pasūtījumi shematisks attēls.

Secīgi izveidosim objektu tipu Piegādātājs un tā objektu tabulu Piegādātāji, šīm nolūkam veicām šādus pierakstus:





Objektu tipa un objektu tabulas shematiskie attēli ir parādīts attēlos 7 un 8.

1. attēls – Objekta tipa Piegādātājs shematisks attēls.

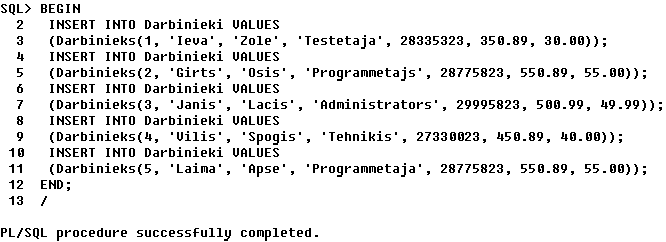
.............

1. attēls – Objekta tabulas Piegādātāji shematisks attēls.

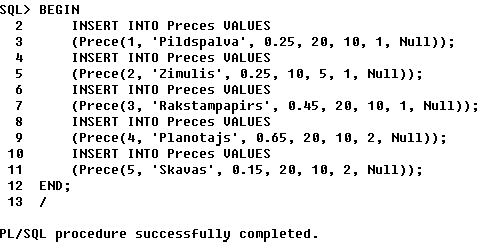
# DATU IEVADE OBJEKTU TABULAS

Pēc datu bāzes struktūras izveidošanas ir nepieciešams aizpildīt objektu tabulas ar datiem, šīm nolūkam izstrādāsim vairākus PL/SQL procedūras ar datu modifikācijas vaicājumiem INSERT INTO.

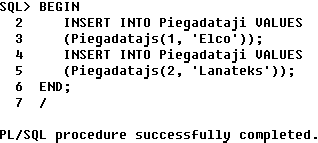
Objektu tabulā Darbinieki piereģistrēsim piecus jaunus darbiniekus, kuri ir indivīdi un katram ir raksturīgas savas īpašības – identifikators, vārds, uzvārds, amats, telefona numurs, alga un prēmija. Lai mēs varētu aizpildīt šo objektu atribūtu vērtības, mums ir nepieciešams noradīt eksemplāra nosaukumu Darbinieks. Darbinieku piereģistrēšanu veiksim ar sekojošo PL/SQL procedūras izpildi:



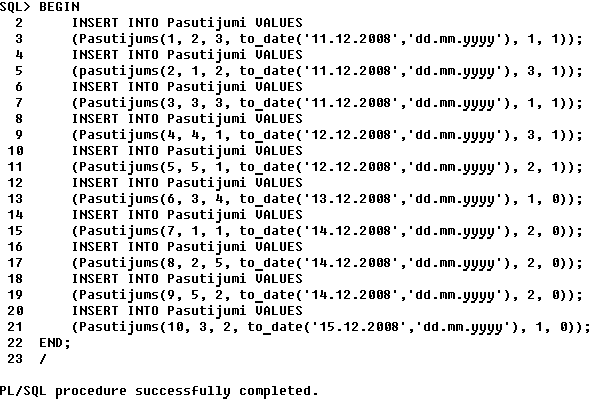
Līdzīgi objekta tabulā Preces veiksim piecu dažādu preču ievadi, norādot to identifikatorus, nosaukumus, cenu, PVN un atlaidi kompānijai procentos, piegādātāja unikālo identifikatoru. Savukārt, kolonnas Maksa lauku vērtībām pagaidām noradīsim nulles vērtības, kuras turpmāk mēs modificēsim, izmantojot šī tipa metodi Gala\_Cena.



Līdzīgi veidā arī veiksim piegādātāju reģistrēšanu, PL/SQL procedūras datu ievades pirmteksts izskatās šādi:



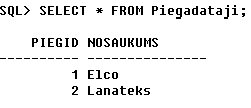
Nākamo aizpildīsim objektu tabulu Pasūtījumi, sastādot PL/SQL procedūru, kurā paredzēs tādu objektu eksemplāru atribūtu vērtību aizpildīšanu kā pasūtījuma, darbinieka un preces unikālie identifikatori, pasūtījuma veikšanas datums, pasūtītas preces skaits un status – vai tā ir piegādāta vai nē (1 vai 0). Šādas procedūras pirmteksts ir sekojošs:



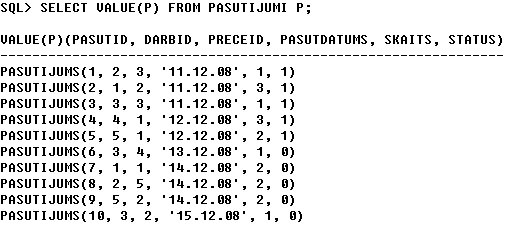
# DATU BĀZES DATU STRUKTŪRAS IZVADĪŠANA

Pēc datu ievades objektu tabulās, pārliecināsimies par datu esamību, konstruējot vairākus OSQL un relāciju SQL vaicājumus, dažbrīd izmantojot punktu notāciju (izgūt datus pēc abām šādam pieejām mēs vairām pateicoties tām, ka Oracle DBVS šādu veidu tabulās objektu atribūtu vērtības ieglabā tieši tabulas kolonnas).

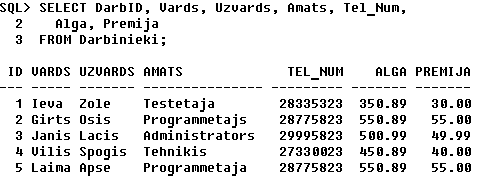
Vispirms konstruēsim parastu datu izgūšanas relāciju SQL vaicājumu, kurš izgūs visus datus no objektu tabulas Piegādātāji, šāda vaicājuma pieraksts un izvades rezultāts ir šāds:



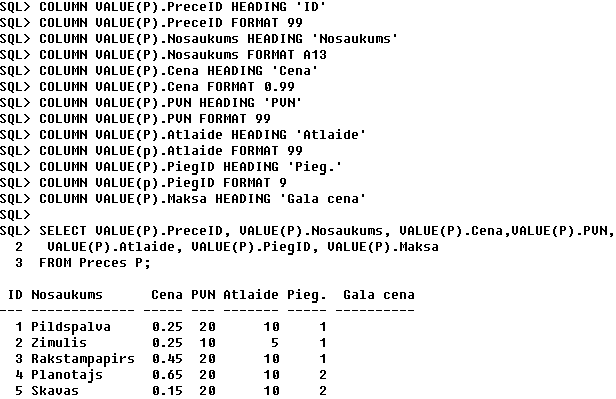
Apskatīsimies objektu tabulas Pasūtījumi datus, konstruējot OSQL vaicājumu, kurā tiks pielietota funkcija VALUE, kura ļauj tieši atsaukties uz objektu eksemplāriem un to atribūtu vērtībām. Vaicājums un tā izvades rezultāts ir sekojoši:



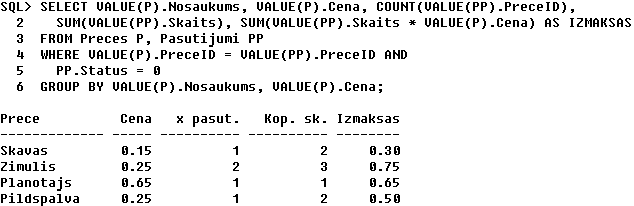
Kā nākamo konstruēsim parastu relāciju SQL vaicājumu, kurš ļaus mums pārliecināties par datu esamību objektu tabulā Darbinieki. Vaicājums un tā izvades rezultāti ir šādi:



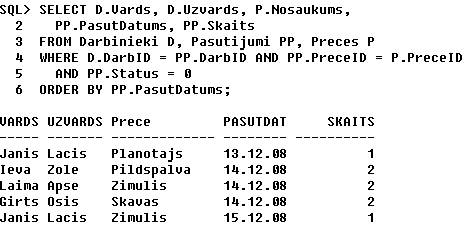
Apskatīsimies objektu tabulas Preces lauku vērtības, konstruēt OSQL vaicājumu, kurā izmantosim funkciju VALUE, lai atsauktos uz objekta Preces atribūtu vērtībām. Vaicājuma pieraksts ir šāds:



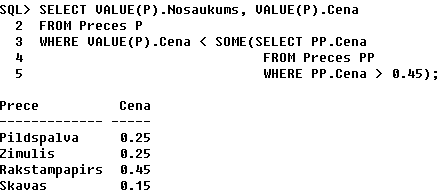
Konstruēsim līdzīgo vaicājumu, kurš izvadīs darbinieku gaidāmo preču nosaukumus (preces, kuras nav vēl piegādātas) un to cenas, paradīs cik reizes dota prece ir pasūtīta (neskaitot tās reizes precēm, kuras ir jau piegādātas), uzradis katras pasūtītas preces kopējo daudzumu un aprēķinās izmaksas. Vaicājuma pieraksts un tā izvades rezultāti ir šādi:



Izstrādāsim vaicājumu, kurš izvadīs informāciju par darbiniekiem, kuri gaida preces, gaidāmo preču nosaukumus un to skaitu, paradis pasūtījuma veikšanas datumu. Šoreiz vaicājumā tiks izmantota punktu notācija. Vaicājums un tā izvades rezultāti izskatās šādi:



Mēs varām arī kombinēt funkcijas VALUE pielietošanu kopā ar punktu notāciju (relāciju SQL pieeju). Šīm piemēram izstrādāsim vaicājumu, kurš atlasīs tikai tādas preces, kuru cenas būs mazākas par tādam precēm, kuru cenas lielākas 0.45 Ls. Vaicājums un tā izpildes rezultāts ir šāds:



# ANONĪMA PL/SQL BLOKA IZVEIDOŠANA

Valodai PL/SQL ir raksturīga blokveida struktūra. Tas nozīme, kā programma PL/SQL sastāv no fragmentiem, kuri var iekļauties viens otrā. PL/SQL bloks var saturēt līdz trim nodaļām:

DECLARE – tajā tiek definēti tipi, mainīgie, konstantes, kursori un citi objekti. Dota bloka nodaļa ir fakultatīva (to var izlaist, programmas rakstīšanas laikā).

BEGIN END – izpildāma nodaļā, tajā tiek realizētas izpildāmās darbības (datu manipulēšanas komandas, SQL operatori, procedūras un funkcijas) . Šī nodaļa ir obligāta iedaļā PL/SQL blokā.

EXCEPTION – izņēmumu nodaļa. Satur aprakstu par to, ka apstrādāt specifiskas kļūdas. Šī nodaļa arī ir fakultatīva.

Valodas PL/SQL bloka struktūra ir sekojoša:

*DECLARE*

*..................*

*BEGIN*

*..................*

*EXCEPTION*

*.................*

*END;*

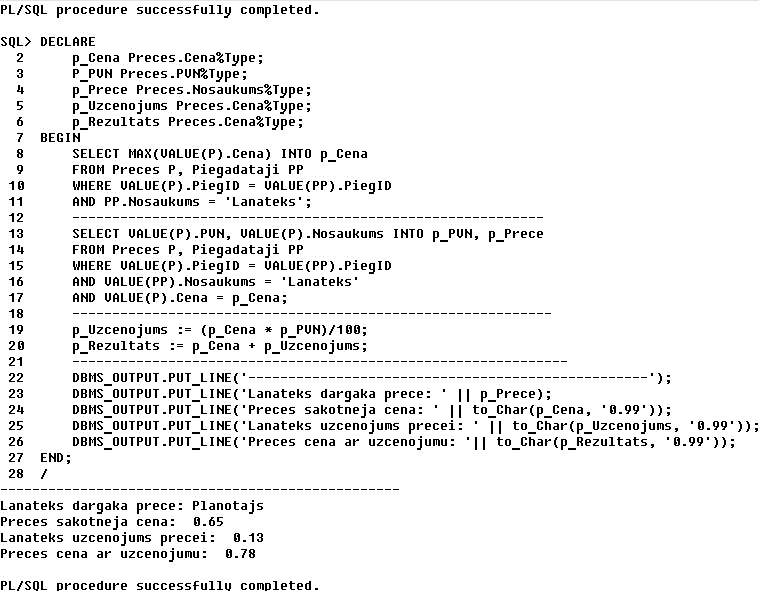
Anonīms PL/SQL bloks ir viens no PL/SQL bloku tipiem, kurš tiek veidots dinamiski un parasti izpildās tikai vienu reizi. Piemēra pēc izstrādāsim nelielu PL/SQL bloku, kurš no objekta tabulas Preces atradis visdārgāko cenu piegādātāja „Lanateks” precei, vadoties pēc kuras turpmāk noteiks tās PVN un nosaukumu, attiecīgi aprēķinās piegādātāja preces uzcenojumu un šīs preces kopējo cenu mūsu kompānijai, kuru sastāda preces sākotnējas cenas vērtības summa kopā ar piegādātāja uzcenojumu.

Šīm nolūkam definēsim piecus mainīgos: p\_Cena – visdārgākās preces cena, p\_PVN – šīs preces PVN, p\_Prece – šīs preces nosaukums, p\_Uzcenojums – mainīgām tiks piešķirta piegādātāja uzcenojuma vērtība, kura tiks aprēķināta preces sākotnējo cenu reizinot ar tās PVN un tad to visu izdalot ar 100, un mainīgā p\_Rezultats tiks saglabāta preces kopēja cenas vērtība. Mainīgiem tiks definēts atbilstošas tabulas kolonnas datu tips ar atribūta %Type palīdzību, šāds paņēmiens ļauj atbrīvoties no datu tipu neatbilstības problēmas programmas izpildes brīdī.

Lai mainīgos ieglabātu attiecīgo objektu tabulas kolonnu lauku vērtības, izmantosim komandu SELECT INTO.

Rezultātu izvadīšanu veiksim ar PL/SQL datu izvadīšanas procedūru DBMS\_OUTPUT. PUT\_LINE palīdzību.

PL/SQL anonīma blokā pirmteksts un tā izpildes rezultāts ir šāds:

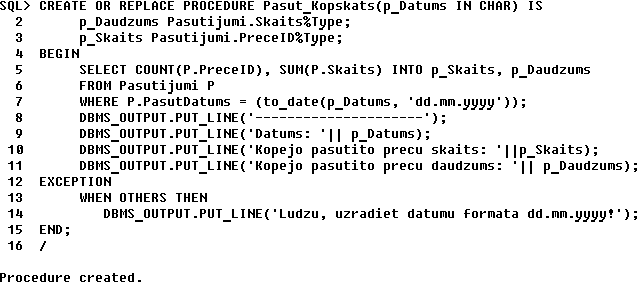


Pēc PL/SQL blokā izpildes rezultātā var redzēt kā visdārgākā piegādātāja „Lanateks” prece ir plānotājs, kuras sākotnēja cena ir 0.65 Ls un piegādātāja uzcenojums šai precei ir 0.13 Ls. Kā rezultātā sanāk kā kopēja preces cena ir 0.78 Ls.

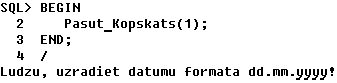
# PROCEDŪRAS IZVEIDOŠANA

Procedūru pamata ir tāds pats PL/SQL bloks. Galvenā procedūru atšķirība ir tāda kā procedūras kompilēta veida tiek glabātas datu bāzē, kas ļauj manāmi uzlabot datu bāzes veiktspēju, ja ir nepieciešams manipulēt ar lielo skaitu datu. Šādi procedūrām arī ir nepieciešams nosaukums. Procedūrām ir iespējams arī nodod parametrus ar argumentu IN, OUT un IN OUT starpību. Argumenti nosaka nodota parametra vērtības modifikācijas iespējās procedūras pirmtekstā, arguments IN nosaka to, ka parametrs būs vienīgi lasāms, savukārt, arguments OUT norāda uz parametra modifikācijas nepieciešamību pirmtekstā un arguments IN OUT atļauj gan modificēt parametra vērtību, gan arī nē. Procedūras definē ar komandas CREATE PROCEDURE palīdzību.

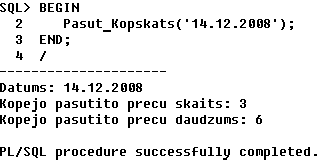
Izveidosim procedūru Pasut\_Kopskats, kurā izvadīs kopējo pasūtīto preču skaitu un daudzumu kāda noradītajā datumā. Šīm nolūkam veicām sekojošas procedūras pierakstu:

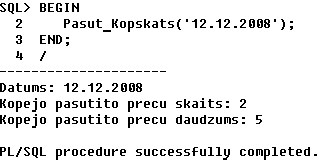


Pēc procedūras pirmteksta var redzēt ka tai tiek nodots parametrs, kurš satur kādu lietotāja norādīto datuma vērtību, vadoties pēc kura šinī datumā vaicājumā tiek saskaitīts un sasummēts pasūtīto preču kopskaits un daudzums. PL/SQL procedūra satur arī izņēmumu nodaļu, kura izvada specifiskas kļūdas paziņojumu, jā lietotājs datuma vietā norāda kādu nekorekto vērtību. Šinī gadījumā vaicājums nevarēs izpildīties un programma paries uz šo izņēmuma nodaļu, kur šo izņēmumu pārķers OTHER izņēmumu kļūdu nosacījums.



Procedūras izsaukumu ietversim PL/SQL anonīmajā blokā. Izsauksim procedūru, nodot tai datumus 14.12.2008 un 12.12.2008.





Rezultātā procedūra izvada mums pasūtīto preču kopsummu un skaitu attiecīgajos datumos.

# PROCEDŪRAS IZVEIDOŠANA, KURA OPERĒ AR KURSORU

Viens no svarīgākajiem procedūrala SQL mehānismiem ir kursori. Kursori ļauj izgūt no datu bāzes rakstu grupu, pārvietoties no viena raksta uz citu un ļauj analizēt tos.

Priekš kursora izmantošanas to vispirms ir jādefinē, jāatver, jāapstrādā ar to rindas, jāaizver un jāiznicina. Kursoru ir iespējams definēt ar DECLARE CURSOR komandu, kuras sintakse ir sekojoša:

*DECLARE CURSOR* kursora\_nosaukums [(parametru\_skaits)] [*RETURN* atgriežamais\_tips] *IS*

vaicājums, ar kura atlasītiem datiem arī operēs kursors.

Kursora atvēršanas brīdī notiek vaicājuma izpilde, uz kura pamatā tiek sastādīta operējamas tabulas struktūra. Kursora atvēršanas sintakse ir sekojoša:

*OPEN* kursora\_nosaukums [(faktisko\_parametru\_saraksts)].

Kārtēja raksta atlasīšanu no kursorā operējamas tabulas rakstiem izpilda komanda FETCH, kuras sintakse ir šāda:

*FETCH* kursora\_nosaukums *INTO* mainīgo\_saraksts;

Bieži vien FETCH komanda tiek iekļauta ciklā, kurš pildās tik ilgi, kamēr komanda FETCH neatlasa visus rakstus no kursora operējamas tabulas, parasti šāda konstrukcija izskatās šādi:

*LOOP*

*FETCH* kursora\_nosaukums *INTO* mainīgais;

*EXIT WHEN* kursora\_nosaukums%*NOTFOUND*;

........- ieraksta apstrāde

*END LOOP;*

Kursora aizvēršana tiek izpildīta ar komandas CLOSE palīdzību, kā rezultāta kursora un viņa operējamās tabulas struktūras dzēšana. Pēc aizvēršanas kursoru ir iespējams atvērt atkārtoti, kursora aizvēršanas sintakse ir šāda:

*CLOSE* kursora\_nosaukums.

Kursoram ir savi atribūti, kuri ļauj noskaidrot kursora tekošo stāvokli. Šādi atribūti ir:

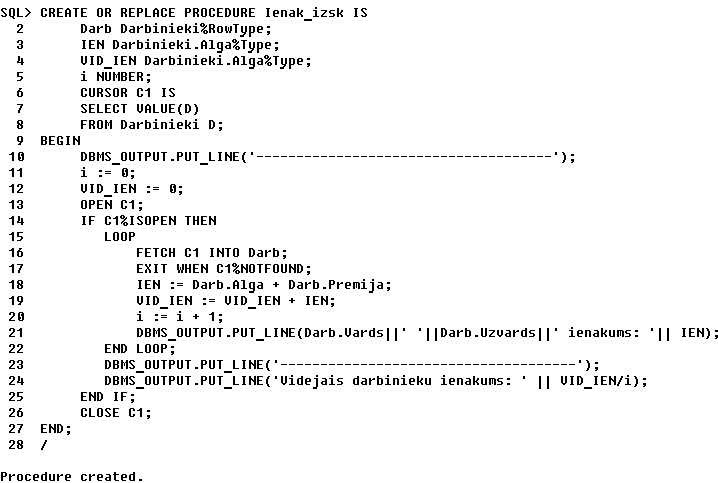
%*ISOPEN* – vai kursors ir atvērts;

%*FOUND* – vai komanda FETCH ir atradusi nākamo rakstu;

%*NOTFOUND* – vai komanda FETCH ir atradusi nākamo rakstu;

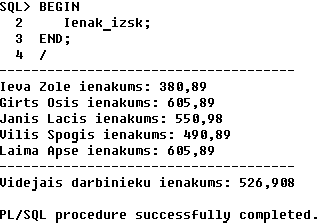
%*ROWCOUNT* – atgriež nolasīto rakstu skaitu.

Izstrādāsim procedūru, kura ar kursora palīdzību saskaitīs kopējos darbinieku ieņēmumus, kurus sastāda algas summa kopā ar prēmiju un aprēķinās darbinieku ieņēmumu vidējos ienākumus, šīs procedūras pirmteksts ir šāds:



Tātad pēc pirmteksta var redzēt, ka tiek definēti četri mainīgie: ien – mainīgā tiek ieglabāta algas un prēmijas kopsummā, vid\_ien – mainīgā tiek ieglabāta kopējo ieņēmumu kopsumma, i – mainīgā tiek ieglabāts kopējais darbinieku skaits, Darb – mainīgā komanda FETCH ieglāba kārtējo rakstu, šīm nolūkam mainīgam tika definēts tabulas raksta datu tips ar atribūta %ROWTYPE palīdzību. Darbinieku tabulas rakstu apstrāde notiek LOOP ciklā, kurā attiecīgi tiek veikti iepriekšminēti aprēķini, pēc cikla izpildes tiek aprēķināta darbinieku vidēja ieņēmumu vērtība.

Pārbaudīsim procedūras darbību izpildot procedūras izsaukumu PL/SQL anonīmajā blokā.

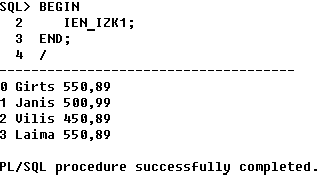


Izstrādāsim procedūru, kurā tiks izmantots asociatīvais masīvs. Asociatīvais masīvs ir datu krātuve, kur katrs raksts ir indeksēts ar kāda tipa mainīgo (skaitlis vai arī simbolu virkne). Atšķirība no iepriekšējas procedūras, šī procedūra izvadīs darbinieku vārdus, kuru alga ir lielāka par 400 Ls. Procedūras pirmteksts ir šāds:



Pēc procedūras pirmteksta ir redzams, ka tanī tiek definēti mas\_Alga un ms\_Vards asociatīvi masīvi, kuru elementu datu tips tiek definēts atbilstoši objektu tabulas Darbinieki kolonnas lauku Alga un Vārds datu tipiem un šie elementi tiks indeksēti ar veselo tipa skaitli. Katrām masīvam tika definēts atbilstošs parametrs, kuri kalpos kā masīvu identifikatori. Lai ieglabātu vai arī piekļūtu pie kāda masīva elementa ir nepieciešams noradīt masīva identifikatoru un indeksu, kā to var redzēt procedūras ciklā ķermenī, kurā atrastos darbiniekus un to algas ieglabā attiecīgos masīvos.

Līdzīgi kā iepriekš, pārbaudīsim procedūras darbību izpildot procedūras izsaukumu PL/SQL anonīmajā blokā.



# FUNKCIJU IZVEIDOŠANA

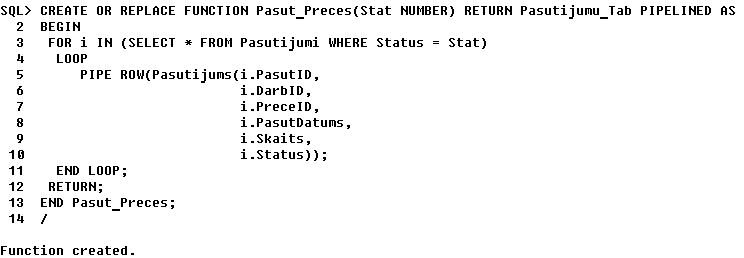
Funkcijas līdzīgi procedūrām datu bāze tiek glabātas kompilēta veidā, tiem ir nosaukumi un tām arī iespējams nodod parametrus, ka arī noradīt to pieejas tiesības ar argumentiem IN, OUT, IN OUT. Funkciju galvenā atšķirība no procedūrām ir tāda kā tās atgriež kādu vērtību.

Izstrādāsim funkciju, kura atgriezis tabulveida struktūru. Funkcijai tiks padots parametrs ar vērtību 1 vai 0, pēc kura vadoties no objektu tabulas Pasūtījumi tiks atlasīta informācija par jau piegādātājiem vai arī vēl gaidāmiem pasūtījumiem. Lai izvadītu šo informāciju tabulveida struktūra, izmantosim PIPLINE pieeju, kurā ļauj funkcijai atgriezt kopējo tabulveida struktūru pa vienai rindiņai.

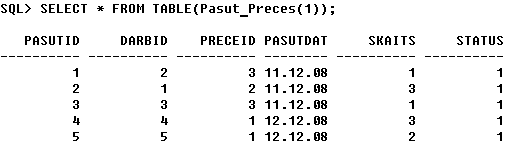
Tātad vispirms mums ir jāizstrādā objekta tipa Pasūtījums kolekcijas tipu, kurš arī būs mūsu PIPLINE funkcijas atgriežamais tips (pamatojoties uz to, katrs atsevišķs objekta eksemplārs tiks iekļauts kopēja kolekcijā un beigu beigās mēs iegūsim tabulveida struktūru).



Nākamo izstrādāsim pašu funkciju, kuras ķermenī ir uzrakstīti divi cikli. For cikls pildās tik ilgi kamēr netiks apstrādātas visas tabulas rindas un šinī ciklā izpildās LOOP cikls, kurā PIPE ROW metode atlasa un atgriež visus rindas elementus.

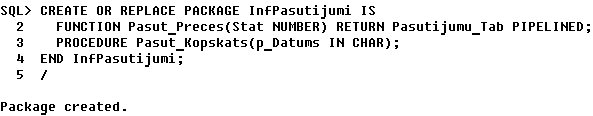


Pašu funkciju mēs varam izsaukt ar datu izgūšanas vaicājumu SELECT ar konstruktora TABLE palīdzību. Procedūras izsaukšana tiek nodemonstrēta nākamā lappusē, kur funkcijai tiek nodots parametrs 1, šādi tiks izvadīta tabula ar informāciju tikai jau par piegādātiem pasūtījumiem.

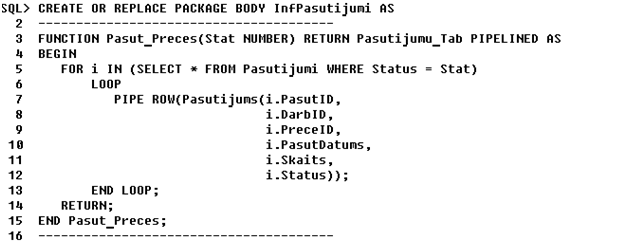


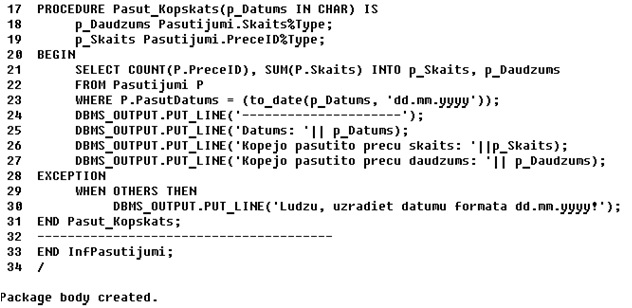
Izstrādāsim programmu paketi. Programmu pakete ļauj sevī apvienot vairākas funkcijas, procedūras un mainīgos. Mūsu pakete ietvers vienu procedūru Pasut\_Kopskats – ļāva apskatīt informāciju par veiktajiem pasūtījumiem kāda datumā un tikot izveidotu funkciju. Ņemot vēra kā dota procedūra un funkcija darbojas ar objektu tabulas Pasūtījumi datiem, paketes nosauksim par InfPasutijumi.

Paketes izveidošanai ir jāveic divas darbības: jāizveido paketes specifikācija un jāizveido tās ķermenī. Paketes specifikācija satur tikai funkciju, procedūru un mainīgo definēšanas daļu. Paketes specifikācijas izveidošanai veicam šādu pierakstu:

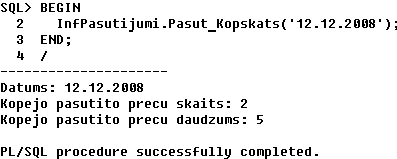


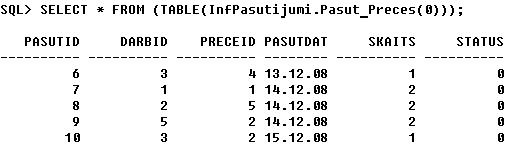
Paketes ķermenis ietver procedūru un funkciju pirmtekstus. Paketes ķermeņa definēšana izskatās šādi.





Lai izsauktu kādu paketes procedūru un/vai funkciju ir jānorāda paketes nosaukums un tad funkcijas vai procedūras nosaukums, vidū lietojot punktu (paketes\_nos.funkcijas\_nos). Izsauksim mūsu paketes procedūru un funkciju, lai pārliecinātos par to darbības pareizību.

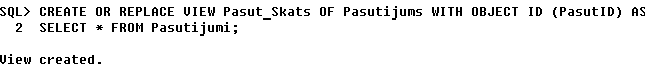




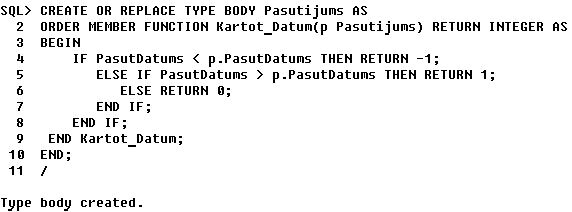
# ORDER TIPA METODES IZVEIDOŠANA OBJEKTU SKATAM

Šajā nodaļā izveidosim objektu tabulas Pasūtījumi objektu skatu un Order tipa metodes ķermenī, kurā sākotnēji tika definēta objektu tipā Pasūtījums. Mūsu uzdevums ir praktiski pielietot šo metodi izveidotājam objektu skatām.

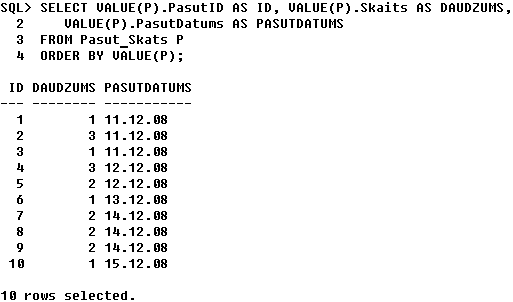
Līdzīgi tām kā relāciju skati ir virtuālas relāciju tabulas, objektu skati ir virtuālas objektu tabulas. Objektu skati satur objektu rindas, to kolonnas sakrīt ar augstāka līmeņa atribūtiem. Objektu skatos objekti tiek saistīti ar šiem skatiem izmantojot kādu identifikatoru, kuru norāda lietotājs frāzē WITH OBJECT ID (specificēts\_identifikators). Objektu skata Pasut\_Skats izveidošanai veiksim šādu pierakstu:

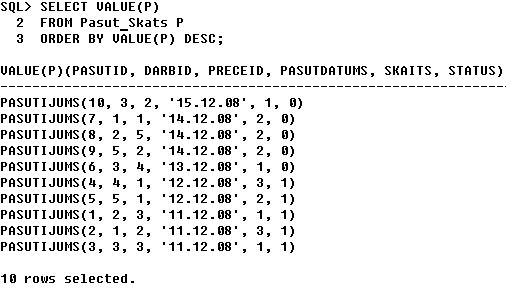
****

ORDER tipa metode izpilda tiešu divu objektu salīdzināšanu, tā salīdzina savu (self) tipa objektu ar funkcijas parametrā noradīto objektu. Metode atgriež negatīvo -1 skaitli, ja parametra objekta vērtība ir lielāka par pašas metodes tipa objekta vērtību, pozitīvo skaitli 1, ja parametra objekta vērtība ir mazāka par pašas metodes tipa objekta vērtību un nulli, ja abu objektu vērtības sakrīt. Mūsu piemēra ORDER tipa metode Kartot\_Datum, noteiks objektu sakārtojumu tabulā, salīdzinot objektu pasūtīšanas datuma atribūtu vērtības. Tā tiks izsaukta automātiski, kā rezultāta ņems tekošo objektu un salīdzinās tā PasutDatums atribūtu ar nākama objekta tādu pašu atribūtu, līdz tiks salīdzināti visi objekti, tad metode ņems nākamo objektu un atkal salīdzinās to ar pārējiem objektiem, līdz netiks sasniegts pilns sakārtojums. Šinī kontekstā -1 viens nozīmes to, ka savs objekts stāvēs zem salīdzināma objekta, 1 virs un 0 paliks sava vietā. Metodes ķermeņa pieraksts izskatās šādi:

****

Veiksim skata objektu eksemplāru sakārtojumu, izsaucot mūsu ORDER tipa metodi, kas tiks panākts ar ORDER BY VALUE(A) kārtošanas komandu. Metodes izpildes rezultāts ir sekojošs:

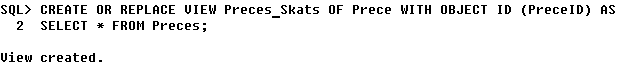
****

****

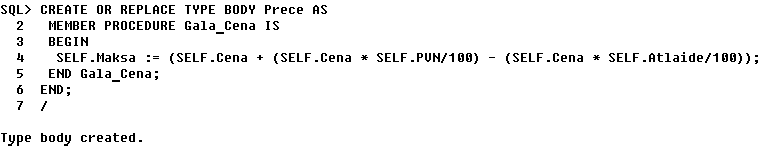
# MEMBER TIPA METODES IZVEIDOŠANA OBJEKTU SKATAM

Šajā nodaļā izveidosim objektu tabulas Preces skatu un Member metodes ķermenī, izstrādāsim PL/SQL bloku, šīs metodes pielietošanai un vaicājumu objektu skatām, metodes izpildes rezultātā attēlošanai.

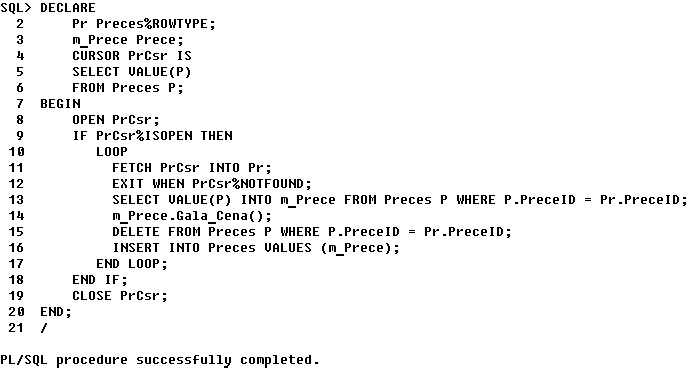
Vispirms izstrādāsim objektu skatu Preces\_Skats uz objektu tabulas Preces pamatā, šīm nolūkam veicām šādu pierakstu:

****

Member tipa metode realizē pieeju pie objekta eksemplāra datiem, to ir jādefinē katrai operācijai, kuru vēlams ar objekta datiem izpildīt. Mūsu gadījuma metode noteiks preces galējo cenu, kuru sastāda pašas cenas summa kopā ar PVN un atņemot Atlaidi. Metode ieglabās galējas cenas vērtību objektu tabulas Maksa kolonnas laukos. Metodes ķermeņa pirmteksts ir šāds:

****

Izveidosim PL/SQL bloku, kurš ļaus mums izsaukt metodi priekš katra objekta tipa.

****

Pēc PL/SQL bloka pirmteksta var redzēt, ka mēs izveidojām speciālu objekta tipa mainīgo m\_Prece, kurām turpmāk LOOP ciklā piešķīrām katru tekošu objekta tipa Prece eksemplāru. Tad katram eksemplāram izsaucām Member metodi, kurā atbilstoša eksemplāra kolonnas laukam Maksa piešķir galējas cenas vērtību. Nākamajās darbības tiek atjaunota pašas objektu tabulas struktūra.

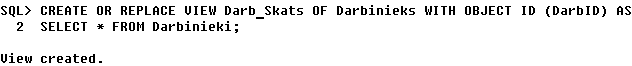
Sastādīsim vaicājumu objektu skatam, kurš paradis Member metodes un PL/SQL bloka izpildes rezultātu. Vaicājuma pirmteksts un tā izvades rezultāts ir sekojošs:

****

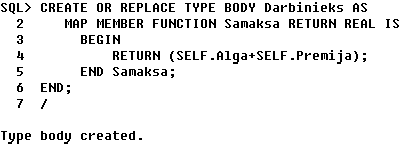
# MAP TIPA METODES IZVEIDOŠANA OBJEKTU SKATAM

Šajā nodaļā izveidosim objektu tabulas Darbinieki skatu un Map tipa metodes ķermenī, praktiski pielietosim šo metodi objektu skatām.

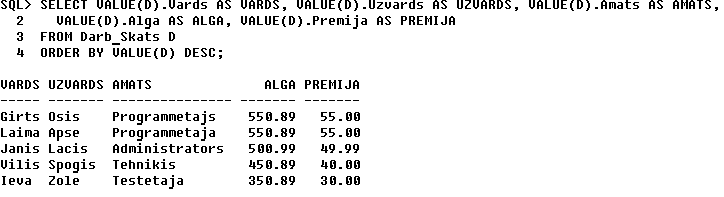
Vispirms izstrādāsim objektu skatu Darb\_Skats, šīm nolūkam veicām šādu pierakstu:



Map tipa metode ir objektu sakārtojuma noteikšanas metode, kas nodrošina pamatu objektu sakārtošanai, salīdzinot objekta eksemplārus, pamatojoties uz kādu no objekta elementiem ar tipu Date, Number, Varchar2 vai to kombinētām izteiksmēm. Metode atgriež vienu no pārskaitītiem datu tipiem. Ja objekta tips definē tādu metodi, tad metode tiek izsaukta automātiski, lai izpildītu tādu salīdzinājumu kā objekts1 > objekts2. Kur objekts1 un objekts2 ir divi objektu mainīgie, kuri var būt salīdzināti izmantojot atbilstības metodi Map(). Izstrādāsim Map tipa metodes Samaksa ķermeni, pati metode ļaus mums sakārtot darbinieku eksemplārus objektu tabulā vai arī objektu skatā pēc to ienākuma lieluma, kuru sastāda algas un prēmijas kopsummā. Šīs metodes pieraksts būs sekojošs:



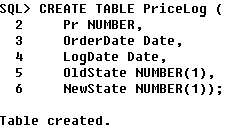
Veiksim objekta skata eksemplāru sakārtojumu, pielietojot attiecīgo metodi, kura līdzīgi ORDER tipa metodei arī tiks izsaukta automātiski.



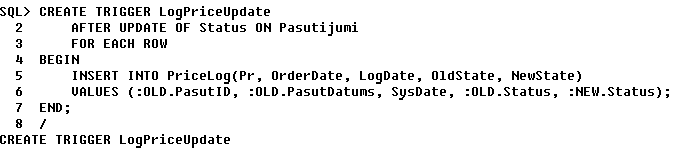
# TRIGERA IZSTRADĀŠANA

Datu bāzes trigeris ir procedūra, kura tiek glabāta datu bāze ka objekts un kuras izpildīšanas tiek inicializēta ar kādu notikumu. Parasti šāds notikums ir saistīts ar datu bāzes modifikāciju, kuru izpilda datu manipulēšanas vaicājumi INSERT, UPDATE un DELETE attiecība pret kādu konkrēto tabulu.

Izstrādāsim nelielu trigeri, kurš reaģēs uz objekta tabulas Pasūtījumi kolonnas lauka Status vērtības izmaiņu (UPDATE), ka rezultātā speciāla žurnālu tabulā ar nosaukumu PriceLog saglabās pasūtījuma identifikatoru, pasūtījuma veikšanas datumu, notikuma notikšanas datumu, veco un jauno lauka Status vērtību. Šādi mēs varam uzzināt, kad lietotājs ir atzīmējis ka viņa noteikta pasūtījuma prece ir jau pienākusi. PriceLog žurnāla tabulas definējums ir šāds:



Trigeru definēšana notiek ar komandu CREATE TRIGGER. Veidojot trigeri ir nepieciešams norādīt tā inicializācijas laiku, parasti, vai nu pēc (after) vai pirms(before) notikuma iestāšanas, notikumu uz kuru reaģēs trigeris, šeit ir trīs iespējas: INSERT (datu ievadīšanu), DELETE (datu dzēšanu) vai UPDATE [OF kolonnas\_lauks] (datu atjaunošanu) un tabulas nosaukumu, kurai trigeris tiks piesaistīts (ON tabulas\_nosaukums). Trigera definēšanas pieraksts ir sekojošs:

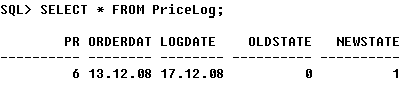


Trigera veidošanas pirmteksts satur tādus modifikatorus, ka :OLD un :NEW, kuri nosaka modificējama objekta vērtību, kura tiks ieglabāta kāda cita tabula, t.i., vai cita tabula tiks ieglabāta lauka veca, vai/un arī jauna vērtība.

Pārbaudīsim trigera darbību, tabulā Pasūtījumi modificējot sēsta pasūtījuma statusu, atzīmējot to ka piegādātu, šīm nolūkam sastādām šādu datu manipulēšanas vaicājumu:



Pēc vaicājumā izpildes pārbaudīsim tabulas PriceLog vērtības, sastādot vienkāršu vaicājumu.



Pēc vaicājumā izvades rezultātiem var secināt kā trigeris ir pilnīgi nostrādājis, attiecīgi mēs varam redzēt pasūtījuma pieteikšanas un pienākšanas datumus, veco un jauno statusu.

# PROCEDŪRAS IZVEIDOŠANA TABULĀM AR REALIZĒTU 1:N OBJEKTU SAITI

Šajā nodaļā izstrādāsim 1:N objektu saiti starp darbiniekiem un to veiktajiem pasūtījumiem (viens darbinieks var veikt vairākus pasūtījumus). Šīm nolūkam izstrādāsim nelielu tabulu Darbinieki2, kura glabās gan darbinieku eksemplārus, gan arī ar iekļūtas tabulas starpību vairākas atsauces uz to veiktajiem pasūtījumiem, t.i., objektu tabulas Pasutijumi2 pasūtījuma eksemplāriem. Šādi mums būs vēl nepieciešams izstrādāt pasūtījumu atsauču objektu kolekcijas tipu. Pēc objektu saites izveidošanas, izstrādāsim arī procedūru, kura operēs ar savstarpēji saistīto tabulu datiem. Objektu saistītas shēma ir paradīta shēma 2.

Darbinieka

eksemplārs

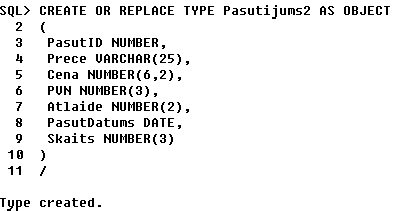
Atsauce (REF)

Pasūtījuma

eksemplārs

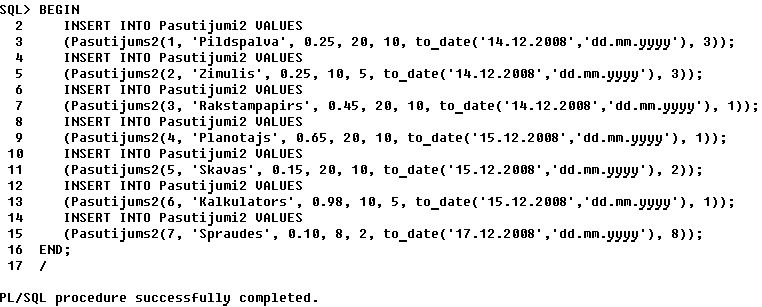
1. shēma – Objektu saistības shēma.

Vispirms izstrādāsim jauno objektu tipu Pasutijums2, kurš saturēs tādus atribūtus kā pasūtījuma unikāls identifikators, pasūtītas preces nosaukums, cena, PVN, atlaide, pasūtījuma veikšanas datums un pasūtītas preces daudzums. Objekta tipa veidošanas pirmteksts ir šāds:

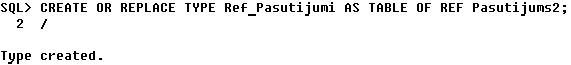


Pasūtījumu objektu tabulas izveidošanai, veiksim šādu pierakstu: 

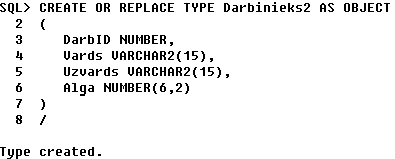
Pēc objektu tabulas izveidošanas, veiksim datu ievadi tajā. Šādi izstrādāsim PL/SQL anonīmo bloku, kurš paredzes septiņu dažādu pasūtījumu pieteikumu reģistrēšanu. PL/SQL anonīma bloka pirmteksts ir sekojošs:



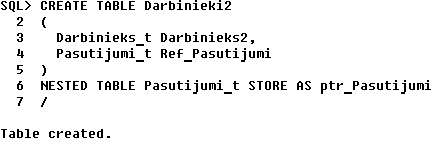
Nākamais, veidojām atsauču objektu kolekcijas tipu, kurš saturēs Pasutijums2 objektu tipu eksemplāru atsauces. Šīm nolūkam veiksim šādu pierakstu:



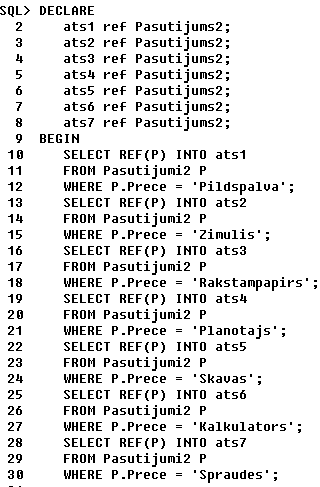
Izveidosim jaunu darbinieku objektu tipu, kurš saturēs tādus atribūtus kā darbinieku identifikatori, vārdi, uzvārdi un algas.

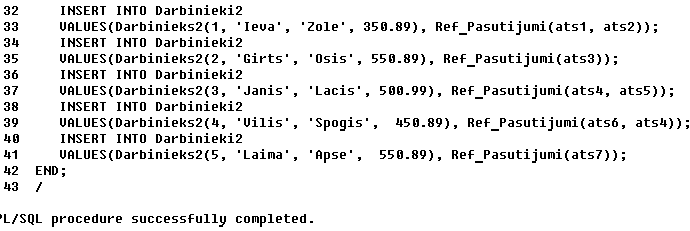


Tagad varam veidot pašu Darbinieki2 tabulu, kurā saturēs kolonnu ar darbinieku objektiem un kolonnu ar atsaucēm uz tabulas Pasutijumi2 objektu eksemplāriem, ar frāzi NESTED TABLE Pasutijumi\_t STORE AS ptr\_Pasutijumi, mēs izveidojam ieliktu tabulu, kura ietvers sevī norādes attiecīgas saistības nodrošināšanai. Tabulas veidošanas pirmteksts ir paradīts nākamā lappusē.

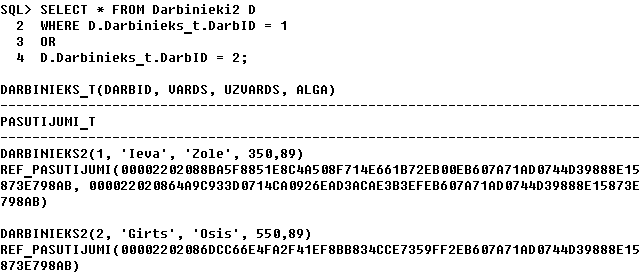


Pēdējais un viss būtiskākais, kas mums atliek, ir veikt tabulas aizpildīšanu, norādot objektu tiešo saistību pēc adresēm. Šīm nolūkam sastādām šādu PL/SQL bloku:

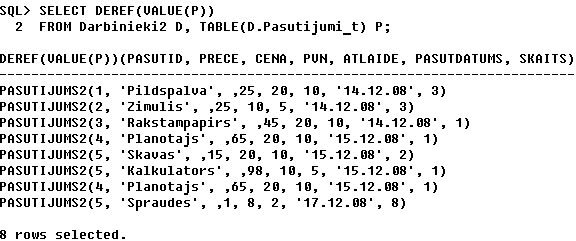




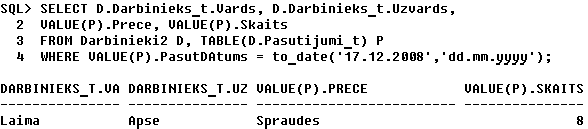
Apskatot tabulas saturu (skat. attēlu zemāk), redzam kā kolonnā Darbinieki\_t tiek glabāti darbinieku objektu eksemplāri un kolonnā Pasutijumi\_t tiek glabāta viena vai vairākas adreses (atsauces), kuras tieša veida uz rāda objektu Pasutijums2 eksemplāriem.



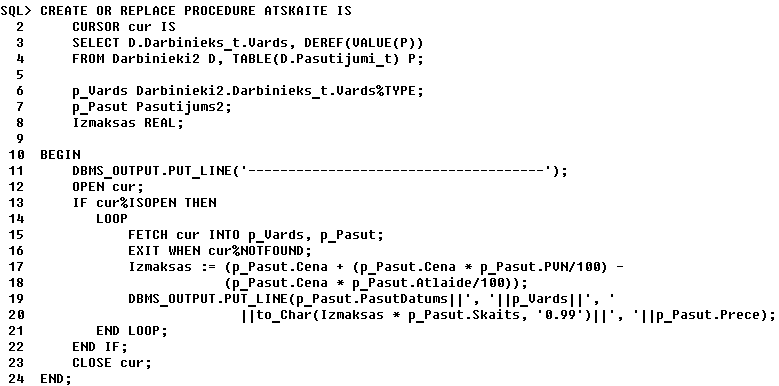
Lai piekļūtu pašiem objektiem pēc to adresēm ir jāizmanto funkciju DEREF, kura ir reversa funkcijai REF (ļauj redzēt objektu adreses pēc objektiem). Mūsu gadījumā vēl papildus ir jālieto TABLE tabulu konstruktoru, kurš no atsauču kolekcijas vērtībām formēs jaunu tabulu, no kuras mēs atlasīsim Pasutijums2 eksemplāru adreses, pielietojot funkcija VALUE. Attiecīgi šīs funkcijas atlasītos rezultātus ir jānodod funkcijai DEREF. Šāds pieraksts izskatās šādi:



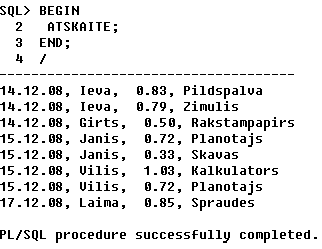
Lai pārliecināties par objektu saistību, konstruēsim šādu vienkāršu vaicājumu:



Izstrādāsim procedūru Atskaite, kura izvadīs darbinieku vārdus, pasūtīto preču nosaukumus un aprēķinās izmaksas, kuru sastāda izvēlētas preces galēja cena, reizinot ar kopējo pasūtītas preces skaitu, attiecīgi pa datumiem. Procedūras pirmteksts ir šāds:



Pēc procedūras pirmteksta var redzēt, ka lai operētu ar objektu pasūtījumu eksemplāru datiem mums bija nepieciešams izveidot attiecīgo šī objekta tipa mainīgo, kurām kursors piešķir tekoša objekta eksemplāra vērtību. Lai pārbaudītu procedūras darbību, izsauksim šo procedūru. Tās izpildes rezultāts ir šāds:



Attiecīgi pa datumiem tiek izvadīts darbinieku un preču saraksts, aprēķināta šī pasūtījuma kopēja izmaksa.

# SECINĀJUMI

Praktiskā darbā galvenais mērķis bija iegūt zināšanas un praktiskās iemaņas darbam ar PL/SQL valodu – PL/SQL anonīmiem blokiem, funkcijām, procedūrām, metodēm, praktiski pielietot un izveidot trigeri, funkciju paketi, izprast kursoru pielietojumus.

Darba laikā tika izpētītas PL/SQL valodas pielietošanas iespējas tādām objektu datu bāzes struktūrām kā objektu tabulas, tipi un skati, tika izpētītās PL/SQL bloku konstrukcijas un valodas operatori, mainīgo datu tipu definēšanas iespējas. PL/SQL procedūrāla valoda sniedz plašākas iespējas dažādo sarežģīto problēmu risināšana, atvieglo datu bāzu uzturēšanu, projektēšanu un administrēšanu. Tā ir izstrādāta galvenokārt tādu problēmu risināšana, kur ar SQL vaicājumu valodu nav pietiekami.

Runājot, par praktiska darbā sastaptajām grūtībām, vairāk laikā atņēma funkciju un procedūru pielaisto kļūdu atklāšana, jo SQL\*PLUS redaktors neizvada procedūru vai funkciju kompilēšanas laikā atklātas kļūdas, tā vietā lietotājam tiek paziņots par funkcijas vai procedūras kļūdaino izveidošanu (Procedure created with compilation errors).

Darbs izrādījās diezgan apjomīgs un prasīgs pēc laika un plānošanas. Vairāk laika tika patērēts tieši dokumentācijas izstrādei.

# IZMANTOTA LITERATŪRA

1. Jānis Eiduks. Lekciju konspekti studiju priekšmetā „Lielās datu bāzes” - Rīga, RTU, 2007.
2. Ян Абрамсон, Майкл С. Эбби, Майкл Кори. «Oracle 10g первое знакомства» - Москва, «Лори», 2007.
3. Филипп Андон, Валерий Резничкою „Язык запросов SQL учебный курс” - Москва, „Питер”, 2006.