Rīgas Tehniska universitāte

Datorzinātnes un Informācijas tehnoloģijas fakultāte

Pirmais praktiskais darbs mācību priekšmetā

„Lielas datu bāzes”

Relāciju datu bāze un SQL paplašinājumi

Izstrādāja: Sergejs Terentjevs

3. kurss, 4.grupa

Apl.nr. 061RDB140

Pārbaudīja: Prof. J. Eiduks

2007./2008. māc. g.

# ANOTĀCIJA

Praktiska darbā izpildes gaita tika izstrādāta relāciju datu bāze, tajā tika paredzēti sekojoši relāciju datu bāzes shēmu objekti: tabulas, klasters, virknes, indekss, skati. Darbam tika izveidots lietotājs un tabulu telpa, praktiski tika pielietoti hierarhiskie, datu noliktas, klona tabulu, datu manipulēšanas un citi datu, meta datu izgūšanas vaicājumi.

Darba pārskatā ir apskatīts praktiskā darba izpildes process, apskatīta datu bāzes arhitektūra, relāciju datu bāzes objekti, SQL paplašinājumi, balstoties uz piemēriem.

Praktiskais darbs tika izpildīts *Oracle 10g Express Edition* datu bāzes vadības sistēmā uz *Genuine Intel* tipā procesorā *Windows XP* operētājsistēmā. Tajā tika izstrādātas 7 tabulas, 2 virknes, 2 skati, 1 klasters, 1 tabulu telpa un 1 lietotājs.

Laboratorijas darba pārskata apjoms: 32 lappuses izklāsta tekstā, tajā ietilpst 17 nodaļas, 3 apakšnodaļas un literatūras saraksts.

# SATURS

[ANOTĀCIJA 2](#_Toc211799268)

[SATURS 3](#_Toc211799269)

[UZDEVUMA NOSTĀDNE 4](#_Toc211799270)

[RELĀCIJU DATU BĀZES UN TABULU PAMATOJUMS 5](#_Toc211799271)

[LIETOTĀJA UN TABULTELPAS IZVIEDOŠANA 8](#_Toc211799272)

[TABULU IZVEIDOŠANA 9](#_Toc211799273)

[VIRKŅU IZVEIDOŠANA 11](#_Toc211799274)

[DATU IEVADE TABULAS 12](#_Toc211799275)

[DATU UN META DATU IZGŪŠANA NO TABULAM 13](#_Toc211799276)

[SKATU IZVEIDOŠANA 17](#_Toc211799277)

[DATU IEVADĪŠANA TABULĀ IZMANTOJOT SKATU 19](#_Toc211799278)

[SKATU DATU UN META DATU IZVADĪŠANA 20](#_Toc211799279)

[KLASTERS 21](#_Toc211799280)

[KLASTERA TABULU DATU UN META DATU IZGŪŠANA 24](#_Toc211799281)

[SQL PAPLAŠINĀJUMI 25](#_Toc211799282)

[Operators CUBE un funkcija GROUPING 25](#_Toc211799283)

[SQL hierarhiski vaicājumi 27](#_Toc211799284)

[Klona tabulas vaicājumi 29](#_Toc211799285)

[SECINĀJUMI 31](#_Toc211799286)

[IZMANTOTA LITERATŪRA 32](#_Toc211799287)

# UZDEVUMA NOSTĀDNE

Izveidot trīs saistītas tabulas:

1. Paredzēt lauku un tabulu ierobežojumus,
2. Datu iestarpināšanu tabulas (8-9rakstiem),
3. Izgūt tabulu meta datus un datus.

Izveidot skatus:

1. Izgūt skatu meta datus un datus.
2. Paredzēt datu ievadīšanu skatos, lai attiecīgi dati tiktu fiksēti attiecīgas tabulas.
3. Paredzēt papildus ierobežojumus.

Izveidot klasteri ar 2 vai 3 tabulām:

1. Izgūt meta datus un datus.

Izveidot divas virknes.

Realizēt SQL paplašinājumus:

1. Izstrādāt 2 datu noliktavas vaicājumus,
2. Izstrādāt 2 klona tabulas vaicājumus,
3. Izstrādāt 2 hierarhiskos vaicājumus.

# RELĀCIJU DATU BĀZES UN TABULU PAMATOJUMS

Praktiskajā uzdevumā tika risināta vietu rezervēšanas problēma avioreisos, uz kuras pamatā tika izstrādāta neliela datu bāze, kas satur trīs relāciju tabulas: Klients, Vieta un Avioreiss.

Tabulā *Klients* tiek fiksēta informācija par klientu, kurš veic vietas vai vairāku vietu rezervēšanu noteiktajā avioreisā. Projektējot doto tabulu, tika nolemts, ka tabulā obligāti jāglabā klienta vārdu, uzvārdu un rēķina numuru – klientam veicot apmaksu, rēķinām tiek piešķirts numurs un tās ir spēkā līdz avioreisa aizlidošanas laikam. Papildus informācija, kuru būtu jāpiefiksē pēc nepieciešamības ir klienta dzimta valsts (pēc noklusējuma LV), pilsēta (pēc noklusējuma Rīga), adrese un telefons. Pēc ievades klientam ir jāpiešķir unikālais identifikators. Tika nolemts, ka klienta vārda un uzvārda maksimālais pieļaujamais simbolu garums ir divdesmit simboli, savukārt, adresei – divdesmit pieci, pilsētai – piecpadsmit, valsts saīsinātam nosaukumam - divi simboli un telefona numuram astoņi numuri. Tabulas shēma un atribūtu apraksts, bez ierobežojumiem un noklusētam vērtībām ir paradīta tabulā 1.

|  |  |  |
| --- | --- | --- |
| **Atribūti** | **Apraksts** | **Datu tips** |
| *Klienta\_num* | Klienta unikālais identifikators | *Number* |
| *Vards* | Reģistrētais klienta vārds | *Varchar2(20)* |
| *Uzvards* | Reģistrētais klienta uzvārds | *Varchar2(20)* |
| *Valsts* | Klienta dzimta valsts | *Varchar2(2)* |
| *Pilseta* | Klienta pilsēta | *Varchar2(15)* |
| *Adrese* | Deklarēta adrese | *Varchar2(25)* |
| *Telefons* | Reģistrētais telefons | *Number(8)* |
| *Rekina\_num* | Rēķina numurs apmaksai, ko veic klients | *Number* |

1. tabula – tabulas *Klients* shēma.

Tabulai ir jāparedz vairākus ierobežojumus, galvenokārt tabulas kolonam *Klienta\_num*, *Vards*, *Uzvards* un *Rekina\_num*. Tātad kolonai *Klienta\_num* izveidosim ierobežojumu ar nosaukumu *K\_Invalid\_PK*, kurš paredzēs, lai kolonas lauku vērtības saturētu unikālos identifikatorus un nepieļautu lauku tukšas vērtības. Kolonām *Vards* un *Uzvards* paredzēsim ierobežojumus ar nosaukumiem *K\_Empty\_Name* un *K\_Empty\_Surname*, kuri nepieļaus, lai kolonu lauki tiktu neaizpildīti. Kolonai *Rekina\_num* izveidosim ierobežojumu ar nosaukumu *K\_Invalid\_CostNum*, kas, savukārt, nepieļaus, lai par rēķina numuru noradītu negatīvo skaitli.

Tabulā *Avioreiss* tiks veikti ieraksti par gaidāmiem avioreisiem uz kādu blakus valsti vai arī tālo, pāri okeānam, valsti. Tāda veidā tabulā būtu jāfiksē vietu skaitu lidmašīnā, lidmašīnas nosaukumu, kura veiks šo avioreisu, valsti un pilsētu jeb pienākšanas punktu, uz kuru dodas lidmašīna, lidmašīnas ierāšanas laiku šajā pilsētā, lidostu, no kuras lidmašīna lido prom un aizlidošanas laiku (skat. tabulu 2). Katrām avioreisam ir jāpiešķir unikālais identifikators.

|  |  |  |
| --- | --- | --- |
| **Atribūti** | **Apraksts** | **Datu tips** |
| *Reisa\_Num* | Avioreisa unikāls identifikators | *Number* |
| *Lidmasina* | Lidmašīnas nosaukums | *Varchar(15)* |
| *Vietu\_SK* | Vietu skaits | *Number(3)* |
| *Valsts* | Avioreisa mērķis | *Varchar(15)* |
| *Pienaksanas\_punkts* | Pienākšanas pilsēta | *Varchar(20)* |
| *Ierasanas\_laiks* | Ierašanas laiks | *Date* |
| *Atiesanas\_lidosta* | Atiešanas lidosta | *Varchar(20)* |
| *Aizlidosanas\_laiks* | Aizlidošanas laiks | *Date* |

1. tabula - tabulas *Avioreiss* shēma.

Līdzīgi iepriekšējai tabulai arī dotas tabulas shēmu ir jāpapildina ar ierobežojumiem un noklusētam vērtībām. Tādejādi kolonai *Reisa\_Num* paredzēsim ierobežojumu ar nosaukumu *A\_Invalid\_PK*, kas nepieļaus kolonas lauku vērtību dublikātus un neaizpildītas vērtības, kolonam *Lidmasina*, *Vietu\_Sk*, *Valsts*, *Pienaksanas\_punkts* izveidosim ierobežojumus ar nosaukumiem *A\_Empty\_Airplane*, *A\_Empty\_PlaceQ*, *A\_Empty\_State*, *A\_Empty\_FlyingPoint*, kuri nepieļaus doto kolonu neaizpildītas lauku vērtības. Kolonam *Ierasanas\_laiks*, *Aizlidosanas\_laiks* izveidosim ierobežojumus ar nosaukumiem *A\_Interval\_ArrivalTime*, *A\_Interval\_FlyAwayTime*, kuri norobežos laika intervālu starp 2008 un 2030 gadu, kuru noradīs lietotājs. Ka pēdējo kolonas *Attiesanas\_lidosta* laukiem noradīsim noklusēto vērtību Rīga.

Nākama tabula *Vieta* glabās informāciju par klienta rezervētu vietu vai vietām noteiktajā avioreisā, šeit lietotājam būs obligāti jānorāda rezervētas vietas numurs, rindas numurs, kurā atrodas dota vieta un maksu par šo vietu (pēc noklusējuma 90 Ls). Bez tām tabulai ir jāsastāv saitē ar divām iepriekš apskatītam tabulām, šīm nolūkam izveidosim laukus *Avio\_Nr* un *Klient\_Nr*, kuri glabās norādes uz unikāliem klientu un avioreisu identifikatoriem(skat. tabulu 3). Katram klienta veiktām pieteikumam (vietas rezervei) ir jāpiešķir unikāls identifikators.

|  |  |  |
| --- | --- | --- |
| **Atribūti** | **Apraksts** | **Datu tips** |
| *ID* | Pieteikuma unikāls identifikators | *Number* |
| *Vietas\_Num* | Rezervētas vietas numurs | *Number(3)* |
| *Rindas\_Num* | Rindas numurs | *Number(3)* |
| *Maksa* | Maksa par rezervēto vietu | *Number(5,2)* |
| *Avio\_Nr* | Norāde uz reisu | *Number* |
| *Klient\_Nr* | Norāde uz klientu | *Number* |

1. tabula - tabulas *Vieta* shēma.

Tabulu *Vieta* ir jāpapildina ar ierobežojumiem *V\_Empty\_PlaceNum* un *V\_Empty\_Line*, kuri neļaus atstāt neaizpildītas kolonu *Vietas\_Num*, *Rindas\_Num* lauku vērtības, papildus paredzēsim ierobežojumu ar nosaukumu *V\_Invalid\_Payment*, kurš neļaus noradīt maksu par rezervēto cenu zemāku par 1 Ls un *V\_Invalid\_PK* ierobežojumu, kurš nepieļaus kolonas *ID* lauku vērtību dublikātus un neaizpildītas vērtības. Tabulu sasaistes kontrolei, kolonam *Klient\_Nr*, *Avio\_Nr* izveidosim ierobežojumus ar nosaukumiem *V\_Klients\_FK* un *V\_Reiss\_FK*, kuri neļaus lietotājam norādīt neeksistējoša avioreisa vai klienta identifikatorus.

Saitei starp tabulām *Klients* un *Vieta* atbilst attiecība *viens pret daudziem*, jo viens klients var rezervēt vairākas vietas viena avioreisā un analoģiski saitei starp tabulām *Avioreiss* un *Vieta* atbilst attiecība *viens pret daudziem*, jo viena avioreisā var ietilpt vairākas rezervētas vietas. Realitāšu saišu diagramma ir paradīta shēma 1.

Avioreiss

Vieta

Klients

Rezervē

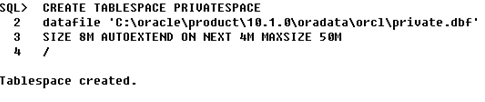
Ietilpst

1. shēma – realitāšu saišu diagramma.

# LIETOTĀJA UN TABULTELPAS IZVIEDOŠANA

*Oracle 10g* datu bāzes uzstādīšanas procesā laikā tiek izveidotas vairākas tabulu telpas (datu bāzes faili), kuras tiek glabāti visdažādākie dati. Tādas tabulu telpas ir *SYSTEM* (glabā datu vārdnīcu un jaunizveidoto objektu datu atribūtus), *SYSAUX* (glabā datu bāzes komponentes un objektus), *UNDO* (glabā izmainītos datus) u.c., paralēli šīm tabulu telpām darba laikā ar datu bāzi tiek izveidotas laicīgas tabulu telpas, piemēram, *TEMPORARY* (laicīgo objektu glabāšanai). Katrām lietotājam ir piesaistīta kāda noteikta tabulu telpa, kurā viņš strādā. Ņemot vēra, ka pēc *Oracle 10g* datu bāzes uzstādīšanas, datu bāzes vadības sistēmapiedāvā strādāt tikai ar lietotāju *SYSTEM* vai *SYS* (pārējie lietotāji netiek aktivēti darbam), man nāktos strādāt *SYSTEM* tabulu telpā. Tāpēc nolēmu vispirms izveidot sev atsevišķu lietotāju un tabulu telpu praktiska darba izpildei.

Šīm nolūkam vispirms autorizējos sistēmā kā *SYSTEM* lietotājs un veicu šādu pierakstu:

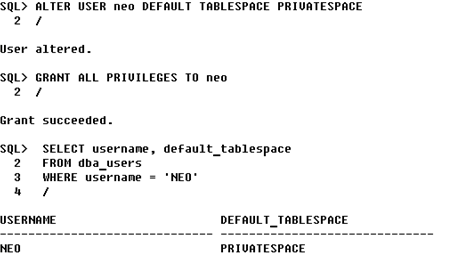


Tādējādi izmantojot „*CREATE TABLESPACE*” komandu, es izveidoju tabulu telpu ar nosaukumu „*privatspace*”, papildus norādot, tabulu telpas datu faila nosaukumu un atrašanas vietu, šī faila sākotnējo izmēru un maksimāli iespējamo izmēru. Datu faila izmērs palielināsies ik pa 4 megabaitiem.

Tad izveidoju lietotāju „*neo*” un norādu tam paroli.

CREATE USER neo IDENTIFIED BY parole

Turpmāk lietotājam piesaistu „*privatspace*” tabulu telpu pēc noklusējuma un piešķiru tām tiesības.



# TABULU IZVEIDOŠANA

Nākama datu bāzes projektēšanas posmā ir jāizveido tabulas, vadoties pēc iepriekš minētam prasībām (skat. nodaļu „relāciju datu bāzes un tabulu pamatojums”). Šīm nolūkam autorizējos sistēma kā „neo” lietotājs un veicu sekojošu pierakstu tabulas *Avioreiss* izveidošanai:

CREATE TABLE AVIOREISS

(

Reisa\_Num NUMBER CONSTRAINT A\_Invalid\_PK PRIMARY KEY,

Lidmasina VARCHAR(15) CONSTRAINT A\_Empty\_Airplane NOT NULL,

Vietu\_SK NUMBER(3) CONSTRAINT A\_Empty\_PlaceQ NOT NULL,

Valsts VARCHAR(15) CONSTRAINT A\_Empty\_State NOT NULL,

Pienaksanas\_punkts VARCHAR2(20) CONSTRAINT A\_Empty\_FlyingPoint NOT NULL,

Ierasanas\_laiks DATE CONSTRAINT A\_Interval\_ArrivalTime

CHECK(Ierasanas\_laiks BETWEEN to\_date('01.01.2008','dd.mm.yyyy') and

to\_date('31.12.2030','dd.mm.yyyy')),

Atiesanas\_lidosta VARCHAR2(20) DEFAULT 'Riga',

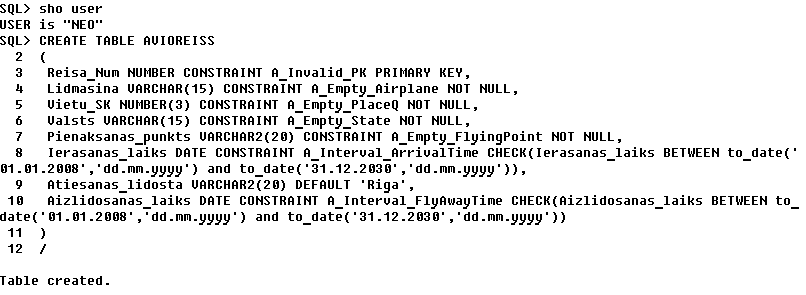
Aizlidosanas\_laiks DATE CONSTRAINT A\_Interval\_FlyAwayTime

CHECK(Aizlidosanas\_laiks BETWEEN to\_date('01.01.2008','dd.mm.yyyy')

and to\_date('31.12.2030','dd.mm.yyyy'))

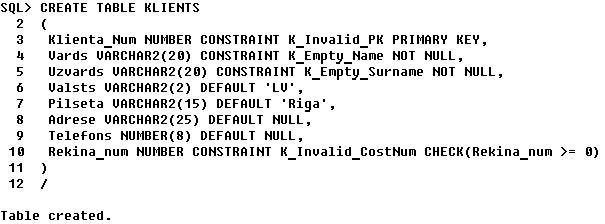
)

Vadoties pēc tabulas struktūras definējuma, redzams, ka tiek izpildīti visi iepriekš pieteiktie nosacījumi tabulai, t.i., tiek definēti visi ierobežojumi un to nosaukumi, ka arī atbilstošam tabulas kolonām tiek piešķirtas noklusētas vērtības (skat. lpp. 6). Ierobežojumu definēšanai vispirms obligāti ir jānorāda atslēgvārds *CONSTRAINT* , savukārt, noklusētas vērtības definēšanai atslēgvārds *DEFAULT*, pie tām ir svarīga definēšanas secīga starp šiem nosacījumiem, t.i., jā tabulas laukam ir nepieciešams definēt gan ierobežojumu, gan norādīt noklusēto vērtību, tad vispirms obligāti ir jānorāda noklusēta vērtība un tikai tad ierobežojums. Ierobežojumiem ar nosaukumiem *A\_Interval\_FlyAwayTime*, *A\_Interval\_Arrivaltime* laika vērtības pārbaudei tiek izmantots operators *BETWEEN*, kurš ļauj salīdzināt kādu vērtību starp diviem ierobežotājiem un funkcija *to\_date*, kurā ļauj pārveidot simbolisko vērtību datumā; laika noradīšanai izmantoju dd.mm.yyyy formātu, kas atbilst pierakstam diena, mēnesis, gads. Lai pārbaudītu kādu vērtību vispirms ir jānorāda *CHECK* atslēgvārds. Pieraksta izpildes rezultāts ir paradīts attēlā 1.



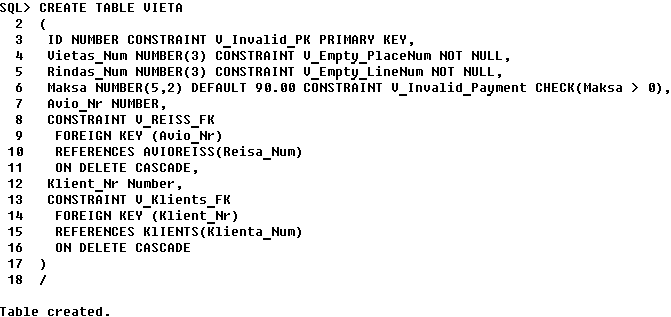
1. att. Tabulas *Avioreiss* veidošana.

Lai izveidotu tabulu *Klients*, veiksim sekojošo pierakstu:



Balstoties uz tabulas struktūras definējumu var redzēt, ka tiek definētas vairākas noklusējuma vērtības un ierobežojumi, piemēram, kolonai *Klienta*\_Num tiek definēts *PRIMARY KEY* ierobežojums, kurš ir atbildīgs par kolonas lauku unikālām vērtībām u.c. ierobežojumi, kuri tika izvirzīti tabulas projektēšanas gaitā (skat. lpp. 5).

Tabulas *Vieta* struktūras definējums:



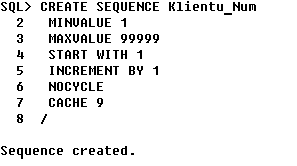
Lai veiktu tabulu sasaistīšanu, izmantoju *FOREIGN KEY* ierobežojumu, kurš norāda uz lauku, kas glabās citas tabulas *UNIQUE* vai *PRIMARY KEY* lauka identifikatorus un *REFERENCES* ierobežojumu, kurš norāda uz citas tabulas *UNIQUE* vai *PRIMARY KEY* lauku, ar kuru veidos saiti.

Papildus norādīju *ON DELETE* *CASCADE* komandu, gadījumam jā no tabulām *Klients* vai *Avioreiss* tiks dzēsts kāds noteikts ieraksts, uz kuru norāda atbilstoša vērtība šinī tabulā, tad šī vērtība tiks dzēsta arī.

# VIRKŅU IZVEIDOŠANA

Pirms datu ievadīšanas tabulās izveidosim virknes, kuras izmantosim primārās atslēgas lauku aizpildīšanai (ģenerēs unikālas vērtības).

Lai izveidotu virkni, SQL\*Plus redaktorā norādām komandu *CREATE SEQUENCE <virnes\_vārds>* ar virkni parametru:



Virknes parametri:

*MINVALUE* – virknes zemākā robeža,

*MAXVALUE* – virknes augšējā robeža,

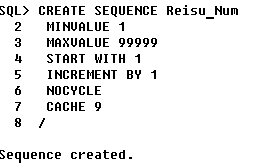
*START WITH* – pirmais virknes ģenerētais numurs,

*INCREMENT BY* – intervāla lielums starp ģenerētiem numuriem (katra nākamā vērtība tiks palielināta par viens),

*NOCYCLE* / *CYCLE* – virknei sasniedzot maksimāli iespējamu vērtību, numuru ģenerēšana sāksies / nesāksies no sākuma.

*CACHE* – veiktspējas uzlabošanai, tiks iepriekš noteiktās un paturētas atmiņā tik vērtības, cik tās norāda lietotājs.

Analoģiski veidojam otru *Reisu\_Num* virkni:



Veicot datu ievadīšanu tabulas, izmantosim virknes, izpildot komandu *<virknes\_vards>.NEXTVAL*, šādi tiks ģenerēts nākamais unikālais numurs primāras atslēgas lauka aizpildīšanai.

Uzzināt virknes nākamo ģenerējamo un patreizējo numuru var ar šādiem vaicājumiem:

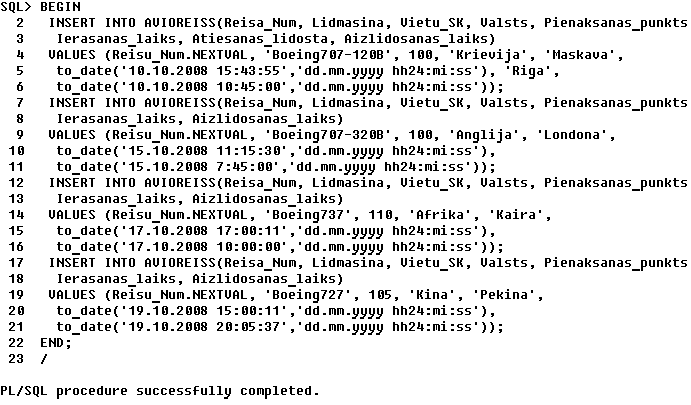
SELECT <sequence\_name>.NEXTVAL FROM dual;

SELECT <sequence\_name>.CURRVAL FROM dual;

# DATU IEVADE TABULAS

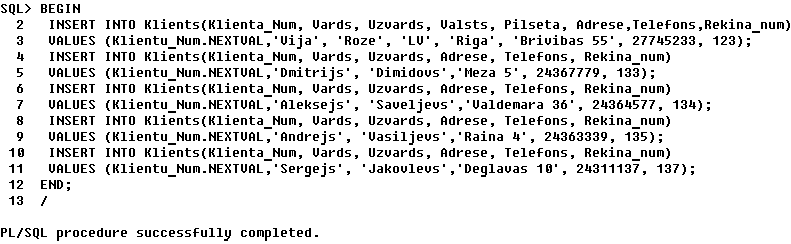
Datu ievadīšanu tabulas veiksim, izmantojot PL/SQL procedūru, ietverot vairākus datu manipulēšanas vaicājumu vienas procedūras blokā (*DECLARE*..*BEGIN*..*EXCEPTION*..*END*).

Sākotnēji veiksim datu iestarpināšanu *Avioreiss* tabulā. Šīm nolūkam *SQL\*Plus* redaktorā ievadām sekojošo pierakstu:

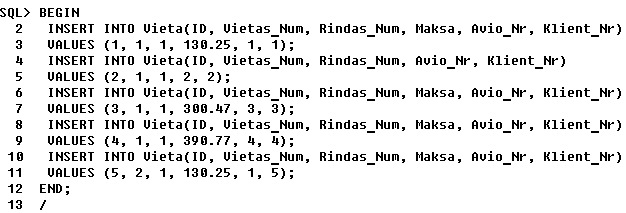


Pēc pieraksta satura var pamanīt, ka datu manipulēšanas vaicājumos *INSERT* kolonas *Atiesanas\_lidosta* lauks netiek uzradīts, izņemot pirmo datu manipulēšanas vaicājumu, šajos gadījumos laukam tiks piešķirta noklusēta vērtība. Pie tam gan datuma, gan laika saglabāšanai, vaicājumos norādu pilno laika formātu dd.mm.yyyy hh24:mi:ss, kur hh24 – stundas, mi – minūtes un ss – sekundes (šeit šo laika vienība varēja arī nenoradīt, jo tā nav obligāta).

Turpmāk ievadīsim datus tabulā *Klients*:



Tagad arī tabulai *Vieta* var veikt datu ievadīšanu (tabulai ir jābūt pēdējai, kurai vadīs ierakstus, citādi var gadīties, ka laukos *Avio\_Nr* vai *Klients\_Nr* tiek ievadīts neeksistējoša klienta vai avioreisa identifikators, ka rezultātā tiks izvadīts sistēmas kļūdas paziņojums un atbilstoša ierobežojuma nosaukums):



# DATU UN META DATU IZGŪŠANA NO TABULAM

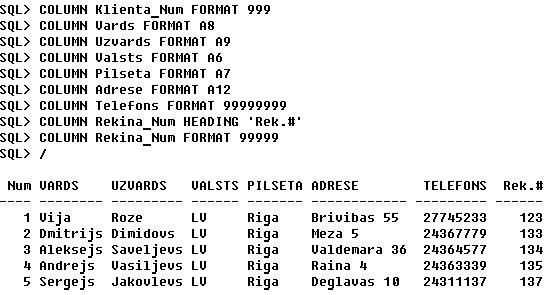
Tagad, kad mēs esam veikuši visas iepriekš nepieciešamas darbības, varam veikt datu izgūšanu no tabulām, lai pārliecinātos par veiksmīgo datu ievadi tabulas. Šīm nolūkam izstrādāsim vairākus *SQL* vaicājumus.

Izstrādāsim vaicājumu, kurš izvadīs visus tabulas *Avioreiss* datus, paredzot vaicājuma rezultāta formatēšanu, lai tas tiktu uzskatamāk attēlots:

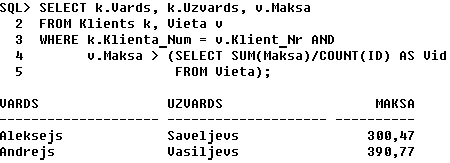


Tātad, izmantojot formatēšanas komandu *COLUMN* un *HEADING* mēs varam izmainīt kolonu nosaukumus virs lauku vērtībām (pēc noklusējuma tiek izvadīti tabulas kolonu nosaukumi); *COLUMN* un *FORMAT* varam noradīt, cik garu skaitli ir jāizvada, decimālzīmes iestarpināšanas vietu skaitli u.tml. (elementi 9 un 0 kalpo kā norādes uz skaitļiem), bez tām varam izmainīt kolonas platumu (pēc noklusējuma kolona ir tik plata, cik simbolus satur kāds lauks, piemēram, laukam ar tipu varchar(16) kolona būs 16 vienības plata) – A*n*, kur *n* ir lietotāja uzdotais platums.

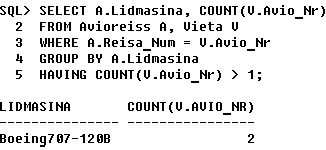
Līdzīgi izgūsim datus no *Klients* tabulas:



Izstrādāsim vaicājumu, kurš atlasa tikai tos klientus, kuru veiktais maksājums ir lielāks par kopējo maksājumu vidējo vērtību (izteiksmes *SUM(Maksa)/COUNT(ID)* vietā varēja arī izmantot agregācijas funkciju *AVG*).

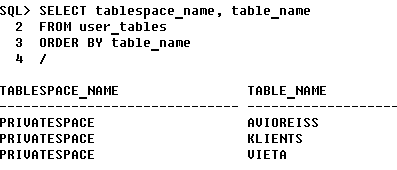


Izstrādāsim vaicājumu, kurš atlasa tikai to lidmašīnu nosaukumus, kurām norīkotu avioreisu skaits pārsniedz 1.

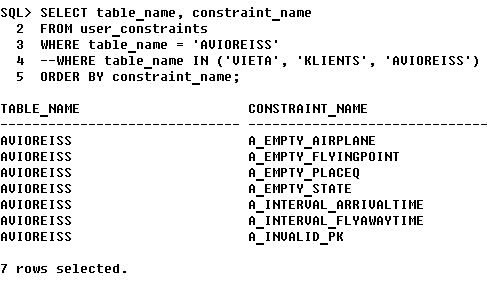


Datu bāzes vārdnīca ietver sevi sistēmas tabulas, kurās glabājas visas datu bāzes izveidotu objektu īpašības un meta dati. Tādas tabulas tiek veidotas automātiski datu bāzes uzstādīšanas laikā (tabulu telpā *SYSTEM*) un to atjaunošana notiek sistemātiski, lietotājam izveidojot, modificējot vai izdzēšot kādu datu bāzes objektu. Datu bāzes vārdnīcas izmantošana ir ļoti lietderīga, jo ļauj lietotājām (atkarība no tiesībām) uzzināt vairāk par datu bāzi, kuru viņš lieto.

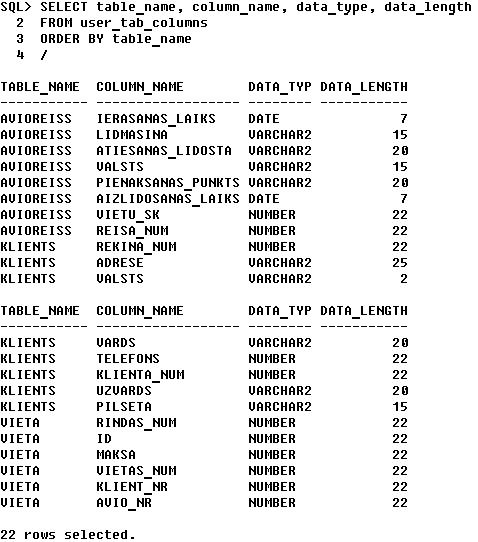
Piemēram, pārliecināsimies kā mūsu tabulas tika izveidotas tabulu telpā „privatspace”, tātad mums ir jāizgūst tabulu un tabulu telpas nosaukumus no lietotāja tabulu skata *user\_tables* (mūs neinteresē citas tabulas no skatiem *all\_tables* un *dba\_tables*) :



Nākamā piemērā izgūsim tabulas *Avioreiss* visus lietotāja definētos ierobežojumu nosaukumus un blakus kolonā izvadīsim tabulas nosaukumu (šoreiz meta datus izgūstam no lietotāja ierobežojumu skata *user\_constraints*) :



Tagad aplūkosim visu lietotāja tabulu nosaukumus, tabulu kolonu nosaukumus, kolonu lauku datu tipus un datu tipu garumus, meta dati izgūsim no lietotāja kolonu skata *user\_tab\_columns*:



# SKATU IZVEIDOŠANA

Tabulas, ar kuram mēs esam darbojušies iepriekš, saucas par bāzes tabulām un tās satur datus. Bez bāzes tabulām eksistē arī loģiskas tabulas – skati, kuri tiek veidoti atlases vaicājuma izpildes rezultātā. Skati no definējuma viedokļa – nosauktais vaicājums, kas tiek saglabāts fiziski un tas tiek katru riezi izpildīts, kad lietotājs vēršas pie tā, bet no izmantošanas viedokļa – parasta tabula. Skatus abstrakti var salīdzināt ar logu, caur kuru skatoties, mēs varam pārskatīt mūs interesējošo informāciju, kurā faktiski glabājas bāzes tabulas.

Skati tiek veidoti norādot komandu *CREATE VIEW*, to veidošanas sintakse izskatās sekojoši:

*CREATE VIEW [OR REPLACE] [[NO] FORCE]* skata\_nosaukums (pseidonīmi) *AS*

Vaicājums uz kura pamatā, tiks veidots skats

[*WITH* [*CASCADE | LOCAL*] *CHECK OPTION* [*CONSTRAINT* < Ierobežojums>]] | [*WITH READ ONLY*] , kur

*OR REPLACE* – parametrs, kas ļauj izveidot skatu no jauna, jā tas eksistē,

*NO FORCE* – parametrs, kurš neļauj izveidot skatu, ja skata bāzes tabulas vēl nav izveidotas,

*FORCE* – parametrs, kurš ļauj izveidot skatu, neskatoties uz to, ka bāzes tabulas vēl nav izveidotas,

*WITH READ ONLY* – nosaka, ka dotais skats netiks pakļauts modifikācijai, t.i., caur to netiks dzēsti, ievadīti un laboti dati,

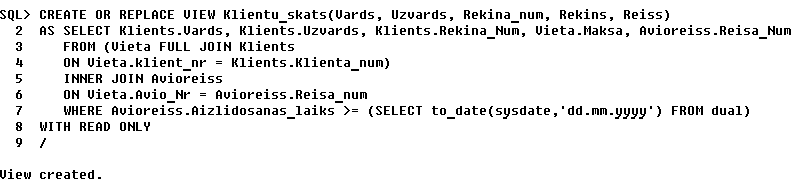
*WITH CHECK OPTION* – parametrs nodrošina datu korekto ievadīšanu un labošanu, balstoties uz skata vaicājuma pamatā, t.i., piemēram, lai nerastos tāda situācija, kad lietotājs caur skatu ievada datus par jaunu klientu, bet pats skats izvada informāciju tikai par kādu noteiktu klientu,

*CONSTRAINST <*ierobežojums*>* - vaicājuma ierobežojums,

*LOCAL* – opcija, kas pārbauda datu integritāti skatā,

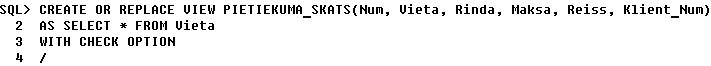
*CASCADE* – opcija, kas ļauj datu integritāti paša skatā un kāda pakārtota skatā.

Izveidosim skatu, kurš ļautu mums efektīvi pārskatīt svarīgāko informāciju par klientiem – vārds, uzvārds, rēķina numurs, maksa par rezervētu vietu un avioreisa numuru, kurā klients ir pierezervējis vietu. Pie tam ir jāatlasa informācija tikai par gaidāmiem reisiem, nevis par tiem, kuri ir jau bijuši.



Pieņemot faktu, ka skats tika veidots vienīgi datu pārskatam, tam tika norādīts *WITH READ ONLY* parametrs.

Izveidosim otru skatu, lai atvieglotu lietotāja darbu, klientu pieteikumu reģistrēšanai (vietu rezervēšanai).



Uzdevuma nostādnē tika minēts, ka ir jāizveido skatu, kas ļautu iestarpināt datus divas vai vairākas tabulas. Šīm nolūkam tika izstrādāts sekojošs skats:

CREATE OR REPLACE VIEW REZERVETAS\_VIETAS(ID, Vards, Uzv, Rekins, V\_ID, VietasNum, RindasNum, Cena, AR\_Nr, K\_Nr)

AS SELECT K.Klienta\_Num, K.Vards, K.Uzvards, K.Rekina\_Num,

V.ID, V.Vietas\_Num, V.Rindas\_Num, V.Maksa, V.Avio\_Nr,

V.Klient\_Nr

FROM Vieta V, Klients K

WHERE V.Klient\_Nr = K.Klienta\_Num

WITH CHECK OPTION

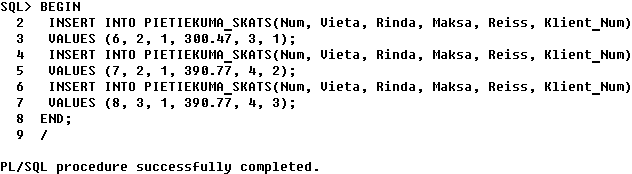
/

Testējot skatu, tika atklāts kā modificēt (*INSERT*, *DELETE*, *UPDATE*) bāzes tabulu datus ar šo skatu nav iespējams, jo sistēma nespēj noteikt kādus ierakstus ir jāmodificē tajā vai citā bāzes tabulā. Iedziļinoties problēmā, atradu svarīgākus kritērijus datu modificēšanas iespējamībai bāzes tabulās ar skatu palīdzību:

* Skatām jābūt formētam no vienas bāzes tabulas,
* Tām nav jāsatur atslēgvārds *DISTINCT* apakš vaicājumā ietvaros,
* Tām nav jāsatur *GROUP BY* un *HAVING* operatorus apakš vaicājumā ietvaros,
* Skata apakš vaicājums nevar satur vēl vienu apakšvaicājumu,
* Lai veiktu datu labošanas un ievadīšanas operācijas, ir jāmodificē visus laukus, kuriem ierobežojums ir *NOT NULL*.

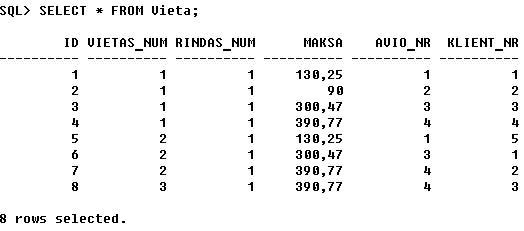
# DATU IEVADĪŠANA TABULĀ IZMANTOJOT SKATU

Sākotnēji tika izveidots *pieteikuma\_skats*, kas ļautu modificēt bāzes tabulas datus. Lai pārbaudītu vai skats ir „modificējams” *SQL \* Plus* redaktorā veicām šādas PL/SQL procedūras pierakstu:



Pēc attēlā var spriest, ka datu iestarpināšana noritēja veiksmīgi, atliek vienīgi patstāvīgi apskatīt tabulas *Vieta* datus.

Datu izvadei no tabulas uz ekrānā, izpildām šādu operāciju:



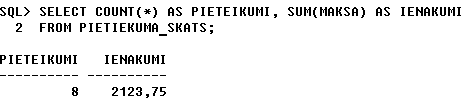
Pēc vaicājuma izpildes rezultātā var pārliecināties par korekta operācijas izpildi.

# SKATU DATU UN META DATU IZVADĪŠANA

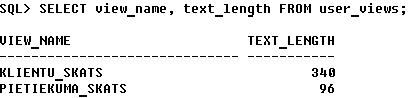
Skatu datu un meta datu izgūšanai izstrādāsim vairākus SQL vaicājumus. Vispirms izgūsim visus datus no *Klientu\_skats* skatā, paredzot attiecīgo kolonu formatēšanu, lai informācija tiktu uzskatamāk izvadīta uz ekrānā (vaicājums ir parādīts zemāk).



Izstrādāsim vaicājumu, kurš saskaita kopējo pieteikumu daudzumu un sasummē klientu veiktus maksājumus.



Izstrādāsim vaicājumu, kurš paredzēs skatu nosaukumu un apakšvaicājuma tekstu garumu izvadi no lietotāja *user\_views* vārdnīcas skatā;



# KLASTERS

Klasters ir objekts, kas satur datus no vienas vai vairākām tabulām, katra no kurām satur vienu vai vairākas kopējas kolonas. Klasterī tabulu raksti tiek turēti blakus un tie glabā vienu un to pašu klastera atslēgu. Šāds princips ļauj uzlabot kopējo ātrdarbību, jo pēc šāda indeksēšanas principa ir iespējams ātrāk piekļūt nepieciešamiem rakstiem. Viena klasterī var ievietot ne vairāk kā 32 tabulas.

Veidojot klasteri un saistot tām tabulas ir visnotaļ svarīgi ievērot darbības secību, t.i., gadījumā jā klasterī tiks veidotas jaunas tabulas, tad būtiski vispirms izveidot klasteri, definēt tajā tabulu struktūras un tikai tad izveidot klastera indeksu, savukārt, ja klasterim ir jāpiesaista jau eksistējošas tabulas ierakstus, tad vispirms ir jāizveido klasters un tā indekss, un tikai tad var veikt jauno tabulu definēšanu klasterī, pamatojoties uz tabulu pastāvošiem datiem.

Klastera izveidošanai ir jānorāda *CREATE CLUSTER* komandu, klastera izveidošanas sintakse izskatās sekojoši:

*CREATE CLUSTER* <klastera nosaukums>

Klastera atslēgas nosaukums un tips

[ *SIZE* ]

[*TABLESPACE*]

[*INDEX*]

[*HASHKEYS*] , kur

*SIZE* - parametrs, kas nosaka izmēru (baitos), lai glabātu visus ierakstus ar vienu un to pašu klastera atslēgas vai heša vērtību,

*TABLESPACE* - parametrs nosaka kāda tabulu telpā klasters tiks izveidots,

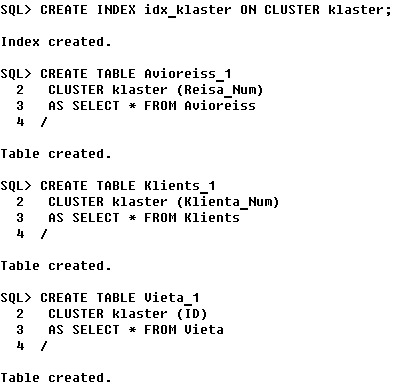
*INDEX* - parametrs nosaka, ka klasteris būs indeksēts,

*HASHKEYS* – parametrs nosaka vai klasters būs heša klasteris, ka arī heš vērtību skaitu katrā heša klasterī.

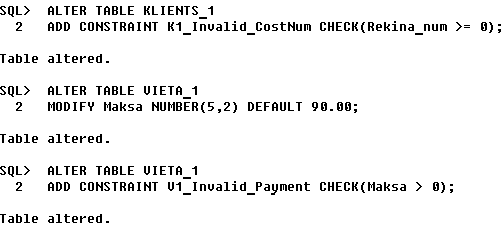
Tātad vispirms izveidosim klasteri ar nosaukumu *Klaster,* klastera atslēgu ar nosaukumu *Klaster\_num*, norādīsim klastera sākotnējo lielumu un vērtību par cik tiks palielināts klastera izmērs.

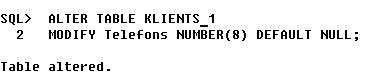


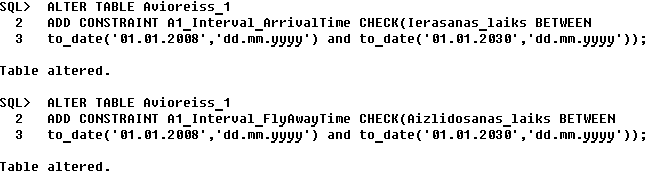
Nākamais, ir jāpiesaista tabulas klasterim; ņemot vēra ka mums ir jau izveidotas tabulas, kas satur datus, tad vispirms izveidosim klastera indeksu un tad veiksim tabulu definēšanu.

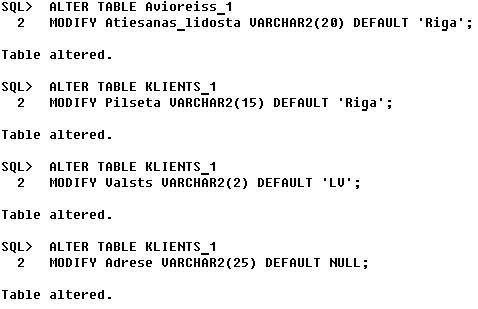


Tagad jaunizveidotajās tabulas *Avioreiss\_1*, *Klients\_1*, *Vieta\_1* jau tiek glabāti tabulu *Avioreiss*, *Klients*, *Vieta* dati. Šeit ir jāpiebilst, ka dotai metodei ir arī savi trūkumi - jaunizveidotas tabulas nesatur iepriekšējo tabulu noklusētas vērtības un ierobežojumus. Lai atrisinātu šo problēmu, mēs varam patstāvīgi pievienot šīs iztrūkstošas komponentes klastera tabulām, izmantojot *ALTER TABLE* komandu.

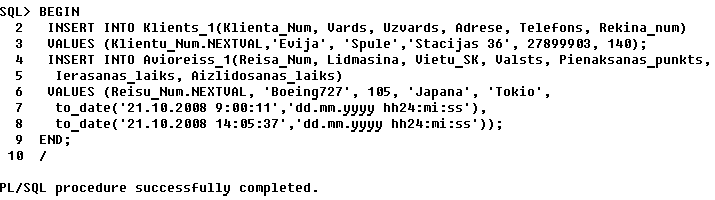


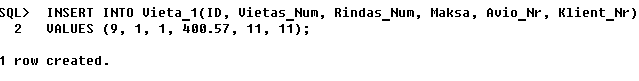






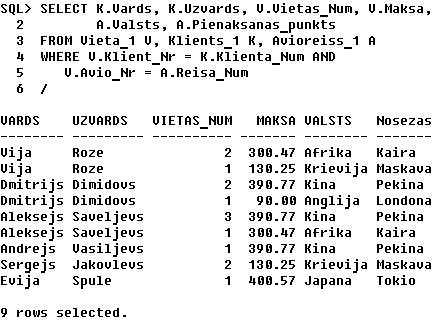
Izveidosim jaunu ierakstus klastera tabulās; tabulai *Klients\_1* pievienosim jaunu klientu Eviju Spūli, tabulai *Avioreiss\_1* pievienosim jaunu riesu uz Japānas pilsētu Tokio un tabulā *Vieta\_1* rezervēsim priekš Evijas vietu avioreisam uz Tokio. Šīm nolūkam izpildām sekojošo *PL/SQL* procedūru un *INSERT* datu manipulēšanas vaicājumu:



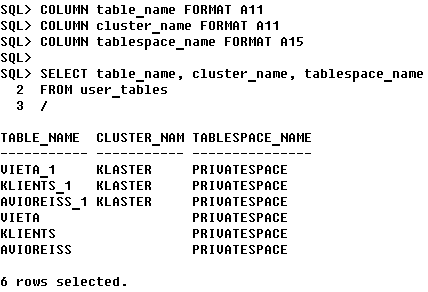


# KLASTERA TABULU DATU UN META DATU IZGŪŠANA

Tagad vienīgi atliek izvadīt informāciju, kuru glabā klastera tabulas. Lai nenāktos vadīt visus datus no visam tabulām, tika izstrādāts SQL vaicājums, kurš atlasa informāciju par klientu, tā vietas numuru un valsti uz kuru viņš dodas (sasaistījām vairākas tabulas). Starp izvadītiem datiem, var redzēt informāciju par nesen reģistrēto klientu Eviju.



Izveidosim vaicājumu, kurš ļaus apskatīt tabulu izvietojumu „*privatespace*” tabulu telpā:



Pēc informācijas, kas tika izvadīta uz ekrānā var redzēt, ka tabulu telpa ir izveidotas sešas tabulas, no kurām trīs ir piesaistītas klasterim.

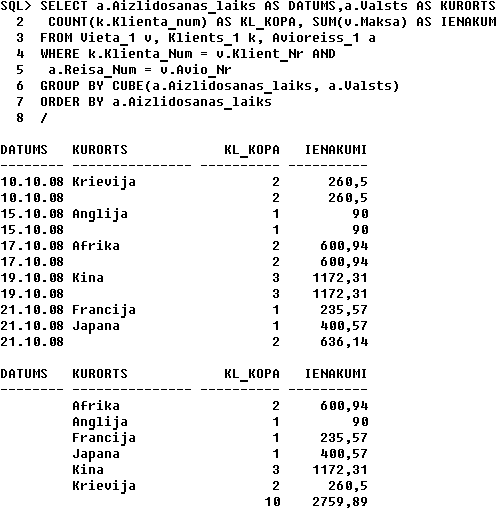
# SQL PAPLAŠINĀJUMI

## Operators CUBE un funkcija GROUPING

Operators *CUBE* ļauj formēt grupas pēc visam tām noradītam dimensiju kombinācijām, tāda veida ļaujot visām šīm dimensijām formēt agregācijas vērtības, t.i., atšķirība no *ROLLUP*, *GROUP BY* grupu operatoriem, *CUBE* ļauj sagrupēt grupās visas tabulu laukus.

Operators *CUBE* tiek lietots komandā *GROUP BY CUBE*, kuras sintakse izskatās sekojoši:

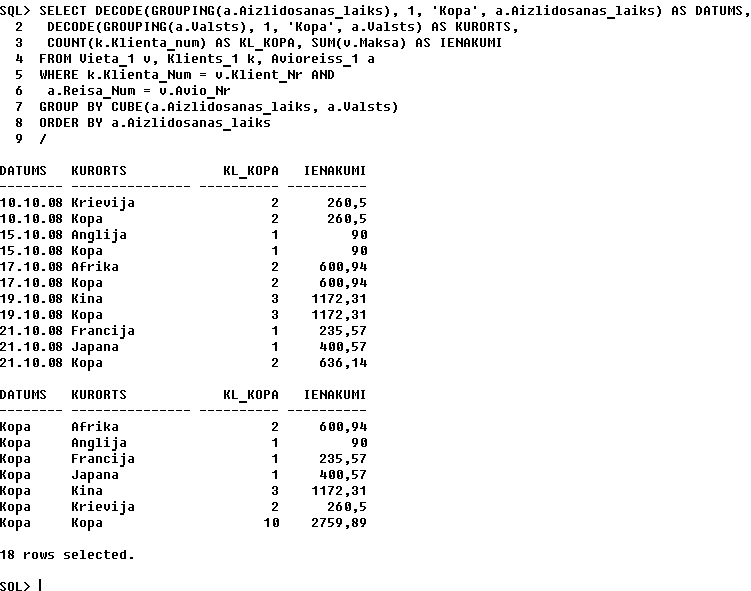
*GROUP BY CUBE* (grupējamo kolonu saraksts). Izstrādāsim tādu vaicājumu.



Pēc vaicājuma izvades rezultātā var spriest, ka sākotnēji tiek apstrādāta *Aizlidosanas\_laiks* dimensija, t.i., tiek veidotas grupas pa datumiem. Šāds paņēmiens, ļauj mums redzēt ieplānotos avioreisus, rezervēto vietu skaitu dotajā avioreisā un kopējo sagaidāmo ienākumu pa datumiem, piemēram, 21. oktobrī ir ieplānoti avioreisi uz Franciju un Japānu, blakus var redzēt kā pagaidām katra avioreisā ir rezervēta tikai viena vietā un kopumā gaidāmais ienākums šajā datumā ir 636.14 Ls. Savukārt, zemāk tiek apstrādāta otra noradīta *Valsts* dimensija - dati tiek sagrupēti pēc avioreisiem, kuriem individuāli tiek uzskaitīts rezervētu vietu skaits un sasummēts kopējais ienākums. Šādi mēs varam spriest, ka pagaidām viss vairāk pieteikumu ir reģistrēti tieši uz Ķīnu un kopējais reisa ienākums ir 1172.31 Ls. Pēdējā rindā tiek saskatīts kopā visu pieteikumu skaits pa avioreisiem un atbilstoši sasummēts kopējais ienākums.

*ROLLUP* un *CUBE* operatori, bez bieži lietojamām agregācijas funkcijā *SUM* un *COUNT*, ļauj izmantot arī citu agregācijas funkciju *GROUPING*. Dota funkcija tiek lietota kā kolonas arguments, kurš pieņem vērtību 1, jā *CUBE* vai *ROLLUP* operators apskatāmajā kolonā izveido vērtību *NULL*, visos pārējos gadījumos tā atgriež vērtību 0.

Par piemēru paņemsim iepriekšējo vaicājumu, tām pievienojot klāt *GROUPING* funkciju kolonu *Aizlidosanas\_laiks* un *Valsts* pārbaudei:



Tātad pēc izvades rezultāta ir redzams, ka kolonu *DATUMS* vai *KURORTS* laukos paradās vārds „kopā”, kas tika paredzēts speciāli - gadījumā jā operators *CUBE*, apstrādājot kolonas *Aizlidosanas\_laiks* vai *Vieta*, atstās lauka tukšo vērtību, funkcija *GROUPING* atgriezis vērtību 1, bet funkcija DECODE pārtvers šo funkcijas izvadi un 1 vietā šajā kolonas laukā izvadīs „kopā”, savukārt, ja funkcija *GROUPING* atgriezis vērtību 0, tad funkcija *DECODE* izvadīs kolonas lauka vērtību, kuru apstrādāja funkcija *CUBE*.

## SQL hierarhiski vaicājumi

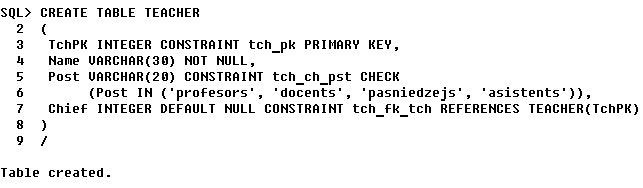
*Oracle* ir paredzēts savs mehānisms, kas ļauj veikt vaicājumu rekursīvo izpildi, t.i., gadījuma jā tabula satur kokveida struktūru, tad rekursīva vaicājuma izpildei ir jānorada hierarhiju līmeņu sakarība - koka augšējo virsotni, kāda veida ir saistīti vecāka un bērna virsotnes. Tas tiek panākts ar šādam divām komandām:

*START WITH* - norāda uz augšējo koka virsotni vai virsotnēm, ar kuru ir jāsāk vaicājuma rekursīvo izpildi,

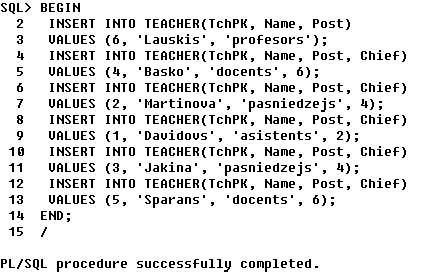
*CONNECT BY* - norāda, kāda veida savstarpēji ir saistīti koka vecāku un bērnu virsotnes; vienkāršāka gadījumā tiek norādīti kādi lauki tiks izmantoti saistības nosacījuma izpildei. Dotajā komanda obligāti ir jānorāda parametrs *PRIOR,* kurš norāda kāds lauks ir vecākais un kāds lauks tām ir pakļautais.

Šeit svarīgi būtu atzīmēt īpašu kolonu *LEVEL*, ar kuras palīdzību tiek panākta tabulas kokveida struktūra (tās atspoguļo lauku hierarhiskos līmeņus). Doto kolonu veido pati sistēma visas lietotāja izveidotājas tabulās, pie tām sistēma pati veic šīs kolonas lauku ierakstus (lietotājam nav atļauts veikt ierakstu šinī kolonā).

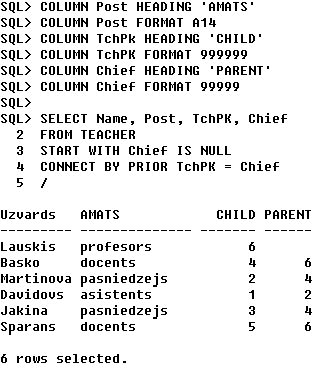
Lai izstrādātu SQL hierarhisko vaicājumu, vispirms izveidosim tabulu, kura saturēs kokveida struktūru. Tabulas sauksies „*Teacher*” (pasniedzējs) un saturēs pasniedzēju unikālos identifikatorus, pasniedzēju uzvārdus, amatu un lauku *Chief* (šefs), kas kalpos kā norāde uz vecāku lauku.



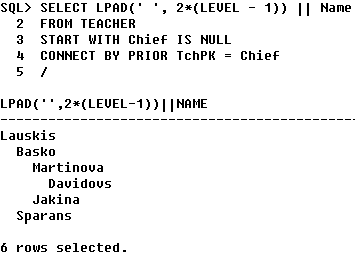
Nākamais, veiksim datu ievadi jaunizveidotajā tabulā:



Tagad varam izveidot hierarhisko vaicājumu, kura uzdevums ir paradīt pasniedzēju hierarhiju, t.i., sameklēt profesoru, docentu, pasniedzēju padotos.

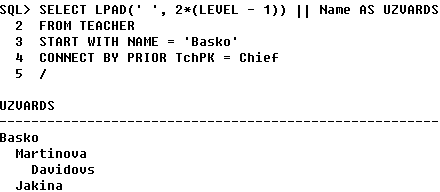


Lai hierarhija tiktu labāk pārskatāmā izmantosim *LPAD* funkciju, kas vadoties pēc vecāko un bērnu saistības, pabīdīs kolonu vērtības.



Tātad pēc vaicājumu izpildes rezultātiem var redzēt, ka profesora Lauska padotie ir docenti Sparāns un Baško, pasniedzēji Martinova un Jakina, asistents Davidovs, savukārt, docentam Baško padotie ir pasniedzēji Martinova un Jakina u.t.t.

*Oracle* dod iespēju attēlot koka kādu apakškoku, šajā gadījuma *START WITH* komandai ir jānorāda apakškoka augšējo virsotni, piemēra pēc atradīsim docenta Baško padotos.



## Klona tabulas vaicājumi

Parasta SQL vaicājumā vienlaicīgi ir iespējams operēt tikai ar vienu ierakstu un tā kolonām, piemēram, var saskaitīt divu kolonu lauku vērtības, sasummēt kolonas lauku vērtības u.t.t., bet tiklīdz rodas nepieciešamība izvadīt citu ierakstu šī paša vaicājuma rezultātā, tad to izdarīt nav iespējams. Protams var to var izdarīt ar kādu no savienojuma veidiem (*join*), bet tās nozīmē krietni sarežģītu datu izgūšanas vaicājumu un tās var iespaidot uz kopējo ātrdarbību (izpildes laiks var palielināties), ka arī var izmantot grupēšanas operatorus un komandas, kas ļaus iegūt grupas lauku summāras, vidējas u.c. vērtības, taču tādā gadījumā tiek zaudēta informācija par ierakstu, no kurienes informācija ir nākusi un nav iespējams iegūt pārējas ieraksta kolonas.

Šīs problēmas risinājumam tika izstrādātas analītiskās funkcijas, kuras ļauj redzēt vērtības no citām rindām un iegūt grupēšanas rezultātus saglabājot arī katra ieraksta specifiskās vērtības. Analītiskas funkcijas balstās uz klona tabulas izveidošanu uz operējamas tabulas pamatā.

Vienkāršota analītisko funkciju sintakse ir šāda:

<funkcija> ( [<arguments>]) *OVER* ([*PARTITION BY* <kolonu saraksts>] *ORDER BY* <kolonu saraksts>), kur

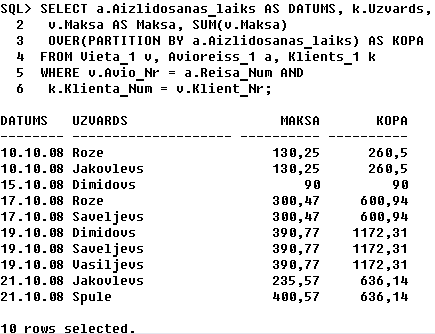
funkcija - konkrētā izmantojamā analītiska funkcija (*MIN*, *MAX*, *SUM*, *RANK* u.c.),

*OVER* – atslēgas vārds, kurš norāda, ka šī ir analītiskā funkcija,

*PARTITION BY* <kolonu saraksts> - nodalījums, kāda tiek sadalīts viss vaicājuma rezultāts,

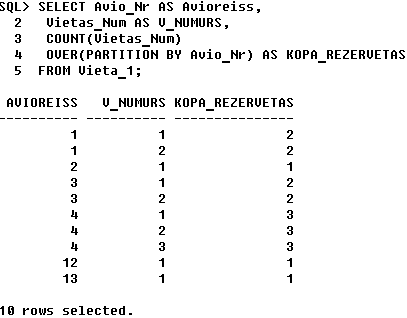
*ORDER BY* - kārtība kādā ieraksti tiks sakārtoti katra nodalījuma ietvaros.

Izstrādāsim divus klona tabulas vaicājumus, lai uzskatāmi attēlotu kāda veidā dotos vaicājumus ir iespējams pielietot praksē. Pirmajā vaicājumā izvadīsim klientu uzvārdus, datumus, kuros klienti ir ieplānojuši doties lidojumā uz kādu valsti, viņu samaksu par rezervētam vietām lidmašīnā un kopējo klientu naudas izdevumus avioreisiem šinīs datumos.



Apskatot vaicājuma izpildes rezultātu var secināt, ka analītiskās funkcijas izmantošana šinī vaicājuma ļāva saglabāt kolonu ierakstu struktūru, neskatoties uz to kā kolonā *KOPA* tika izmantots grupēšanas paņēmiens.

Otrā vaicājumā izvadīsim avioreisu unikālos identifikatorus, rezervētas vietas un kopējo rezervēto vietu skaitu dotajos avioreisos.



# SECINĀJUMI

Praktiskā darbā galvenais mērķis bija iegūt zināšanas un praktiskās iemaņas darbam ar relāciju datu bāzes shēmas objektiem - tabulām, indeksiem, skatiem, virknēm un datu bāzes arhitektūru kopumā - tabulu telpām, datu vārdnīcas skatiem, shēmām, iepazīties ar Oracle datu bāzes paplašinātājam iespējam datu izgūšanai un organizēšanai tabulas.

Darba laikā tika izprasts, cik ļoti svarīgi ir veikt šo shēmas objektu korekto organizāciju tā, lai nodrošina adekvātu datu bāzes veiktspēju un tanī paša laikā neaizņemtu pārāk daudz vietas uz fiziskā diska (relāciju shēmu objekti ir cieši saistīti ar tabulas telpas un datu faila izmēriem). Papildus iemācījos praktiski pielietot SQL paplašinājumus, kurus var uzskatīt par SQL vaicājumu valodas jaunievedumiem, kuri nesen tika iekļauti kopēja SQL standartā. Šīs fakts norāda uz SQL vaicājumu valodas attīstību, kas, savukārt, nevar neiepriecināt (SQL vaicājumu valoda tiek piemērota vēl sarežģītāko uzdevumu risināšanai).

Runājot, par praktiska darbā sastaptajām grūtībām, vairāk laikā tika patērēts skata izstrādei, kurš ļautu veikt datu ievadi vairākas tabulas - tieši problēmas iemesla atrašanai, kāpēc sistēma neļauj to izdarīt, otrkārt, darbā sākuma bija jānolemj vai klasteri veidot uzreiz vai arī vadoties pēc uzdevuma nostādnes, darba vidū.

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

Runājot, par Oracle datu bāzes rīkiem un lietojumiem, kas nāk kopā ar datu bāzi, tie spēj pilnīgi apmierināt datu bāzes administratora mijiedarbību ar datu bāzi. Šeit ir jāpiebilst, ka Oracle ir labi piemērots apmācībai, t.i., šeit administratoram visu ir jāveic patstāvīgi, kas rosina lielāku izpratni par datu bāzēm kopumā, atšķirība no MSSQL datu bāzes, kurai līdzi nāk jau grafiskais interfeiss, kas ļauj atvieglot administratora rutīnu. Oracle datu bāze rosina papildus interesi šādā ziņa, ka to ir iespējams lietot vairākas operētājsistēmas līdzīgi MySQL, kas nav tik funkcionāla salīdzinot ar Oracle datu bāzi. Sākotnēji man bija vēlēšanas uzstādīt Oracle datu bāzi uz Linux operētājsistēmas, bet šeit man nācās sarūgtināties, jo kompānijas Oracle projektā „Unbreakable Linux” neietilpst mans iecienītākais Linux distributīvs.

.

# 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.