Latvijas Universitāte
Fizikas un matemātikas fakultāte
Datorikas nodaļa

Migrācijas ceļi no un uz ORACLE

Bakalaura darbs

Autors
Irina Gerasimenko

Vadītājs
Juris Strods

Dr. sc. comp.
Asoc. profesors

Rīga, 2006

Anotācija

Bakalaura darba mērķis ir izpētīt un salīdzināt Datu Bāzes Pārvaldības Sistēmas (DBPS) migrācijas iespējas, problēmas un to risinājumus. Tika izpētīts migrācijas process no Adabas D uz Oracle un no Oracle uz Adabas D DBPS. Darbā ir aprakstītas Adabas D un Oracle DBPS un to SQL programmēšanas valodas. Galvenais uzsvars ir likts uz datu bāzes shēmas un biznesa loģikas migrāciju.

Annotation

The purpose of this bachelor work was to study and compare Database Management Systems (DBMS) migration possibilities, problems and solutions. Migration process had been studied on Oracle and Adabas D DBMS. In this work were described Adabas D and Oracle DBMS and their SQL programming language. Main accent was taken for database scheme and business logic migrations.

Аннотация

Целью бакалавской работы является изучение и сравнение возможностей миграции Систем Управления Базами Данных (СУБД), проблем связанных с миграцией и их решений. Процесс миграции был изучен на Oracle и Adabas D СУБД. Были описаны Oracle и Adabas D СУБД и их SQL языки программирования. Более углублённо описана миграцию схемы и бизнесс логики баз данных.

Autoreferāts

Autore strādā par programmētāju uzturēšanas projektā, kas ir realizēts Oracle vidē. Darbā ir izpētītas un salīdzinātas Datu Bāzes Pārvaldības Sistēmas (DBPS) migrācijas iespējas no Adabas D uz Oracle un no Oracle uz Adabas D DBPS, problēmas un to risinājumi.

Darba izstrādes laikā tika izpētīts migrācijas jēdziens un tās organizēšana. Tika identificētas problēmas, kas var rasties migrācijas laikā un kādi risinājumi šīm problēmām patreiz eksistē.

Tika noskaidrots kā vēl nav neviena migrācijas rīka, kas realizē migrāciju no un uz Adabas D.

Darbā ir izpētīta Adabas D DBPS struktūra un tās SQL programmēšanas valoda - SQL-PL. Līdzīgi Adabas D ir aprakstīta arī Oracle SQL programmēšanas valoda PL/SQL.

Darbā ir uzskaitītas migrācijas fāzes, parādītas problēmas un to risinājumi.

Saturs

1 Ievads

Bakalaura darba tēmas izvēlei par pamatu bija nepieciešamība izpētīt Adabas D migrācijas iespēju uz DBPS, kas nodrošinātu analogu funkcionalitāti un dotu papildus iespējas.

Pastāv divi biežāk sastopamie iemesli, kāpēc nepieciešams veikt migrāciju:

1. programmatūra ir novecojusi, un ir radusies nepieciešamība pāriet uz jaunāku versiju. Šajā gadījumā jaunāka programmatūras versija nodrošina nepieciešamo funkcionalitāti.
2. organizācijai kaut kādu iemeslu dēļ ir nepieciešams pāriet uz pavisam citu atšķirīgu programmatūru. Programmatūrai nav pieejamas jaunas versijas vai tā nenodrošina nepieciešamo funkcionalitāti.

Ar datu bāzēm ir tāpat kā ar programmatūras migrāciju, t.i. nav jēgas uzturēt datu bāzi ar ierobežotam iespējam (piemēram, Adabas D), ja ir līdzīga DBPS (piemēram, Oracle), kas ļauj tai pašai datu bāzei izvērsties nepieciešamos mērogos, un šāds risinājums ir finansiāli pieņemams. No otras puses, ja, piemēram, JAVA projekts ir uztaisīts uz Oracle datu bāzes, taču nekādas specifiskas Oracle DBPS iespējas netiek izmantotas un uzturēt tādu datu bāzi ir dārgi, tad konkrētā organizācija var izvēlēties pāriet uz DBPS (piemēram, Adabas D), kas spēj vajadzīgo funkcionalitāti nodrošināt.

Šī darba mērķis ir izpētīt problēmas, ar kurām ir jāsaskaras organizācijām pārejot uz dārgāku un jaudīgāku vai arī otrādāk, uz lētāku datu bāzi. Darbā ir uzskaitītas problēmas, ar kurām nāksies sastapties, kam ir jāpievērš uzmanība, lai migrācija notiktu pareizi. Ir aprakstīti iespējamie problēmu risinājumi.

Lai sasniegtu mērķi autore ir atsevišķi izpētījusi Oracle un Adabas D DBPS, un migrācijas procesu no Adabas D DBPS uz Oracle un no Oracle uz Adabas D.

Darbā ir aprakstīts, kas notiek katrā migrācijas fāzē, parādītas problēmas, kas pastāv konkrētās migrācijās, un kā tās var atrisināt vai apiet.

Ņemot vērā to, ka darba autorei ir pieredze ar Oracle DBPS, kā otra DBPS ir izvēlēta Oracle. Oracle DBPS principā ir spējīga aizvietot Adabas D DBPS.

Šis bakalaura darbs sastāv no četrām nodaļām.

Pirmajā nodaļā ir izklāstīts migrācijas jēdziens, dots ieskāts kā jānotiek migrācijai, aprakstītas virspusējas datu bāzes migrācijas problēmas un pašreiz pieejamie šo problēmu risinājumi.

Otrajā nodaļā ir izpētītas un īsi aprakstītas pašas Oracle un Adabas D datu bāzes pārvaldības sistēmas.

Trešajā nodaļā ir aprakstīts migrācijas process no Adabas D uz Oracle, ir pārskaitīti iespējamie ieguvumi un zaudējumi no šādas migrācijas. Pa soļiem aprakstīts, kādas un kur ir jāveic pārejas izmaiņas un beigās veikti secinājumi par šādu migrāciju.

Ceturtā nodaļa veidota līdzīgi trešajai, tajā ir aprakstīts migrācijas process no Oracle uz Adabas D.

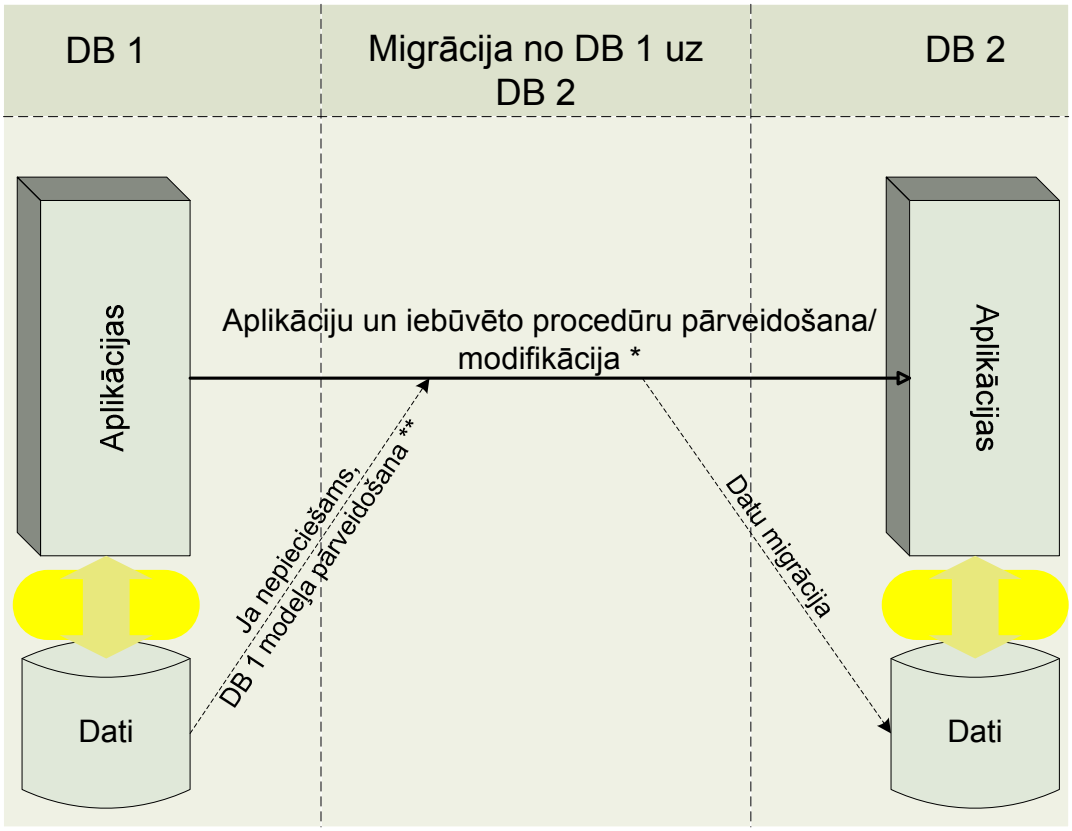
Darbā beigās ir izdarīti secinājumi un aprakstīts kas tika panākts ar šo darbu.

2 Migrācija

2.1 Kas ir migrācija?

Migrācija - pāreja no vienas aparatūras vai programmatūras sistēmas uz citu sistēmu tā, lai esošos lietojumus un datus varētu izmantot citā datorā vai vidē ar citu operētājsistēmu [1].

Es savā bakalaura darbā izskatīju migrāciju no vienas datu bāzes pārvaldības sistēmas (turpmāk tekstā vienkārši datu bāze) uz otru. Tātad datu bāzes migrācija ir pāreja no vienas datu bāzes uz citu datu bāzi tā, lai saglabātos gan iepriekšējas datu bāzes funkcionalitāte, gan dati (Zīm. 1).



Zīm. 1 - vizuālais datu bāzes migrācijas attēlojums

* nozīmē, ka pārtaisa visas datu bāzes funkcijas, procedūras, trigerus, aplikācijas u.t.t.

** nozīmē, ka ja mēs, piemēram, gribam migrēt no datu bāzes ar hierarhisko vai objektu modeli uz datu bāzi, kura šādu modeli neatbalsta, tad ir jāpārveido arī modelis, uz to, kādu atbalsta jauna datu bāze un kas ir piemērotāks projekta prasībām.

Programmatūras pārveidošanu/modificēšanu un datu pārnešanu var veikt manuāli, t.i. ņemt katru funkciju/procedūru un atsevišķi pārveidot to otrai datu bāzei atbilstošā formātā. Tā darīt, protams, var, taču tam vajag pārāk daudz laika un resursu, un ja datu bāze nav primitīva, tad noteikti tā darīt nevajag, jo iepriekšējas datu bāzes biznesa loģika un aplikāciju funkcionalitāte iespējams atšķirsies no iegūtajiem, bet mūsu mērķis ir pārnes datu bāzi ar visu to funkcionalitāti, nevis modificēt to.

Var rīkoties savādāk, izmantojot vai pašrocīgi uzrakstīt/pārveidot rīku, kas varēs automātiski veikt aplikāciju, iebūvēto procedūru un datu migrāciju. Tā var ieekonomēt gan laiku, gan resursus, gan pieļaut mazāk kļūdu.

Principā ir divi datu bāzes migrācijas veidi – pāriešana uz jaunāko versiju (upgrade) un pāriešana uz citu datu bāzi. Parasti pāriešana uz jaunāko datu bāzes versiju ir vieglāka atšķirībā no pāriešanas uz citu datu bāzi, jo jaunas datu bāzes versijas bieži vien atšķiras no vecam tikai ar jauno pieliktu klāt funkcionalitāti un nebūtiskam datu bāzes struktūras izmaiņām, un pat ja kaut kas no datu bāzes struktūras tiek izmainīts, tad tas ir labi aprakstīts jaunās versijas dokumentācijā. Dažas datu bāzes pat piedāvā dokumentāciju ar detalizētu migrācijas aprakstu. Savukārt, lai pāriet uz citu datu bāzi, vai nu ir firmai jātaisa rīks, ar ko vārēs veikt automātisko migrāciju, vai arī ir jāpērk tāds rīks (protams, ja tāds vispār eksistē).

2.2 Kāpēc ir vajadzīga migrācija?

Tikko sistēma izaug, paradās vairākas problēmas ar veco datu bāzi, piemēram, vajadzība datu bāzes „palielināšanai", vairs nav iespējams optimizēt biznesa procesus, ir pārāk zema veiktspēja, aplikāciju neatbilstība realitātei un citas. Tas samazina kopēju sistēmas produktivitāti un viens no veidiem kā ar to cīnīties ir atrast sev atbilstošāku datu bāzi un migrēt uz to.

Tipiskie iemesli, kāpēc tiek nolemts migrēt, ir:

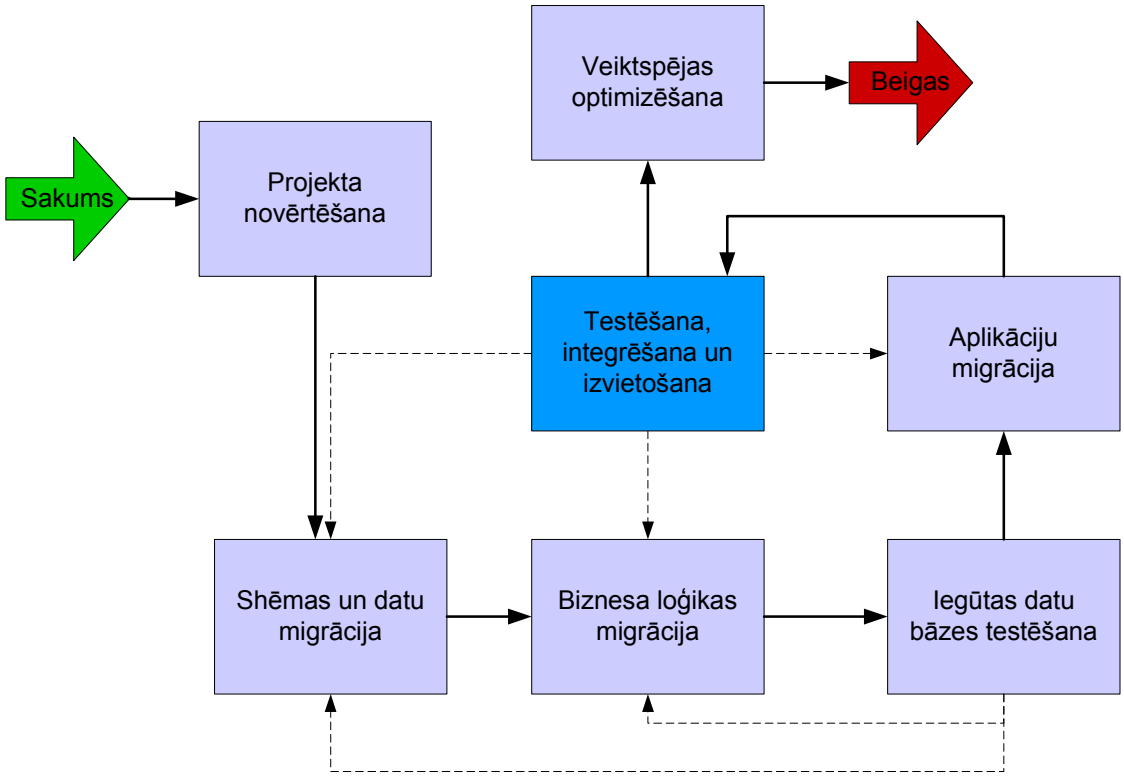
1. pārsniegts datu lielums;
2. pārsniegts lietotāju skaists;
3. organizācijas kopējie izdevumi/ienākumi (uz DB izmantošanu un administrēšanu);
4. sistēma izmanto vairākas DB un ir problēmas ar to savstarpējo darbību.

Parasti datu bāzi izvēlas produktīvāku, kurai piemīt drošība un vairākas izvēršanas iespējas. Tāpēc migrācija kļūst par efektīvāku veidu, lai samazinātu sarežģītību un uzlabot sistēmas produktivitāti, ka arī saglābāt naudu, nepārrakstot visu DB funkcionalitāti [2].

Taču migrācija nav nemaz tik viegls un vienkāršs process! Jo migrējot no vienas datu bāzes uz otru paradās vairākas problēmas, bet par tām sīkāk nākamajā nodaļā.

2.3 Kā notiek migrācija

Lai uzskatamāk paradīt migrāciju kā procesu šajā nodaļā ir aprakstītas galvenās fāzes datu bāzes migrācijai, kā arī tās ir paradītas zīmējumā Zīm. 2. [3, 4, 5]



Zīm. 2 – datu bāzes migrācijas dzīves cikla modelis

1. Projekta novērtēšana

   Pats pirmais kas ir jāizpēta - cik liels ir projekts, lai zinātu cik daudz laika, resursu un naudas vajadzēs tā migrācijai uz citu datu bāzi. Bieži gadās, ka pēc šis izpētes organizācijas saprot, ka projekta pārnešanai uz citu datu bāzi aizies pārāk daudz laika un naudas, un ja ir acīmredzami organizācijas zaudējumi pat pēc projekta pārnešanas un iedarbināšanas uz citas datu bāzes, tad tiek nolemts migrāciju neveikt.

   Rodas jautājums, no kā var spriest cik daudz laika, resursu un naudas tiks patērēti projekta migrācijai. Patiesība tas nav triviāls jautājums, jo projekts ir pamatīgi jāizpēta, par to uz ko ir jāskatās ir aprakstīts nodaļā *2.4.1 Kam ir jāpievērš uzmanība...* Kad ir zināms, kas būs jāmaina konkrētajā projekta, tad var arī rēķināt cik uz to aizies laika, resursu un naudas. Un ja tomēr šie „cipari" būs apmierinoši, tad sākas pati migrācija.

2. Shēmas un datu migrācija

   Shēma ir pamats jaunas datu bāzes definīcijai. Migrējot uz pavisam citu datu bieži var gadīties tā, ka kaut kas no shēmas detaļām vai pat viss datu bāzes modelis netiek atbalstīts jaunajā datu bāzē. Tādā gadījumā shēma būs jāpārtaisa un tas nozīme, ka nebūs vienkārši datu kopēšana no vecam tabulām uz jaunam, jo tabulu struktūras var ļoti atšķirties no vecajām! Un vēl ir jāņem vēra, ka datu kodējumi dažādās datu bāzes var atšķirties, šis jautājums ir sāpīgāks datu bāzēm, kas neatbalsta UNICODE. Jo sabojāt datus reāli strādājošam projektam būtu pārāk sāpīgi.

3. Biznesa loģikas migrācija

   Šeit notiek funkciju, procedūru un trigeru migrācija. Šis process ir daudzas reizes sarežģītāks nekā shēmas un datu migrācija, jo nav nevienas datu bāzes ar pilnīgi identiskam procedūru valodām (stored procedure language) un tas nozīme, ka šajā posmā vajadzēs visu pārveidot. Parasti ir tā, ka vienas datu bāzes procedūras valodas izmantotās īpatnības (features) nemaz nav atbalstītas otras datu bāzes procedūras valodā, bet, ja pat kaut kādas īpatnības atbalsta abas datu bāzes procedūru valodas, tad noteikti tām ir atšķirīga semantika un sintakse. T.i., ja kāda funkcija netiek atbalstīta otrā datu bāzē, tad to būs jāizveido, ja ir līdzīga, tad to būs attiecīgi jāmodificē.

4. Iegūtās datu bāzes testēšana

   Tikko ir izveidota jauna datu bāze un ir pārnesti dati un kods, ir jāpārliecinās, ka jaunā datu bāze atbilst vecajai. Lai tā tas būtu, visiem operācijas rezultātiem ir jābūt identiskiem ar veco datu bāzi. Tas nozīme, ka jālaiž visas operācijas abās datu bāzēs un pēc izpildes jāpārliecinās par rezultātiem. Pēc veiksmīgas datu bāzes testēšanas var uzskatīt, ka datu bāze ir migrēta un gatava izmantošanai.

5. Aplikāciju migrācija

Projekti parasti sastāv no aplikācijām (applications), kas izmanto datu bāzi, piemēram, padodot tai SQL vaicājumus un izsaucot datu bāzes iekšējas funkcijas un procedūras. Tas nozīme, ka visas vietas, kur ir kaut kāda aplikāciju komunikācija ar datu bāzi ir jāpārtaisa, lai projekts strādātu ar jauno datu bāzi tāpat kā ar veco, jo datu bāzes shēma visticamāk ir jau savādāka, nekā vecajā datu bāzē un iespējams kaut kādas iekšējas procedūras/funkcijas ir pārsauktas.

Protams, ja šis aplikācijas izmanto standarta draiverus ODBC vai JDBC un SQL vaicājumi nav dinamiski, tad šī stadija būs diezgan viegla, jo būs vienkārši jāatrod vietas, kur ir šie vaicājumi un ir jāpārtaisa tie atbilstoši jaunās datu bāzes shēmai. Savādāk būs, ja SQL vaicājumi ir dinamiski. Tāda gadījumā būs jāpārskata viss kods, kur vaicājumi varētu veidoties, jāizanalizē tos un jāpārraksta. Un ja aplikācijas ir diezgan lielas, tas varētu aizņemt diezgan daudz laika un ir risks, ka aplikācija vairs nestrādās kā agrāk.

Pavisam savādāk būs, ja aplikācijas izmanto savu datu bāzes lietojumprogrammas saskarni (API) vai ir rakstītas ar savdabīgu valodu. Tādā gadījumā būs vēlams vai nepieciešams pārrakstīt pilnība aplikācijas. Un, ja notiks tāda veidā, tad šis migrācijas posms varētu aizņemt laika vairāk, nekā visi iepriekšēji, t.i. lai to visu uzrakstītu un pēc tam attiecīgi notestētu.

6. Testēšana, integrēšana un izvietošana
   Šai fāzei nevajadzētu aizņemt pārāk daudz laika, jo ir pieņemts, ka datu bāze mums jau ir izveidota, notestēta un akceptēta, ir tikai jānotestē vai aplikācijas strādā ar to tāpat kā iepriekš. T.i., jāpārliecinās vai aplikācijas ir pareizi modificētas.

7. Veiktspējas optimizēšana
   Tikko viss strādā korekti ir vēlams noregulēt datu bāzi tā, lai uzlabotos veiktspēja. Tātad ir jānoskaidro jaunas datu bāzes ātrdarbības uzlabošanas

algoritmi un tikko tas ir noskaidrots var mēģināt pielabot procedūras, funkcijas un SQL vaicājumus, lai tie strādātu ātrāk. Var pat izrādīties, ka ir jāpārtaisa indeksu struktūra vai savādāk fiziski jāuztur datu bāze.

2.4 Migrācijas problēmas

Tā kā migrācija nav nemaz tik primitīvs un vienkāršs process un migrējot no vienas datu bāzes uz otru parasti nākas sastapties ar vairākām problēmām, ir nolemts atsevišķi pārskaitīt kam uzreiz plānojot migrāciju ir jāpievērš uzmanība un kādas tad tieši eksistē migrācijas problēmas.

2.4.1 Kam ir jāpievērš uzmanība...

Lai zināt cik daudz un ko tieši būs jāpārtaisa ir vērts apskatīties sekojošo [2, 4]:

a. Vai patreizējās datu bāzes modelis var tikt pārnests *neizmainīts* uz citu datu bāzi, t.i. vai otrā datu bāze atbalsta to pašu modeli. Protams, patreiz vairākums datu bāzes atbalsta tikai relācijas modeli (Relational model), taču nedrīkst aizmirst, ka eksistē arī hierarhiskais modelis (Hierarchical model), tīkla modelis (Network model), dimensiju modelis (Dimensional model) un objektu datu bāzes modelis (Object database model), kā arī eksistē datu bāzes, kuras strādā tikai ar šādiem modeļiem [6].

b. Vai datu bāzes tabulas var tikt pārnestas nemainītas uz citu datu bāzi, t.i. vai var izmantot tos pašus vai līdzīgus datu tipus, vai datu bāze atbalsta primāras un ārējas atslēgas, vai var uztaisīt unikalitātes (unique constraint) un pārbaudes (check constraint) ierobežojumus, vai ir iespējams taisīts indeksus, lauku komentārus un citas ar tabulām saistītas lietas.

c. Vai datu bāze atbalsta to pašu SQL valodas versiju kā patreizēja datu bāze, ja nē, tad noteikti būs jāpārtaisa visus vaicājumus (parastie SQL teikumi, dinamiskie SQL teikumi un skatījumi).

d. Vai pārnesot datus tos nevares sabojāt, t.i., ja patreizēja datu bāze atbalsta UNICODE, vai to atbalstīs arī datu bāze, uz kuru notiks migrēšana.

e. Jāizpēta cik ļoti atšķiras „SQL programmēšana valoda". Noteikti katrai datu bāzei tā būs savādāka, taču ir jāzina kādas un cik daudz funkcijas ir atbalstītas abās datu bāzēs un, ja kaut kas nav atbalstīts vienā vai otrā datu bāzē, tad vai un cik grūti ir tas aizvietošana/modificēšana.

2.4.2 Virspusēji par migrācijas problēmām

a. Datu tipi

   Dažādās datu bāzēs eksistē līdzīgi un atšķirīgi datu tipi un, lai saglabātu datus obligāti ir jākonvertē datus uz attiecīgiem datu tipiem. Jau iepriekš ir jānolemj ko darīt šajā sakarā, vai atrast līdzīgu datu tipu un to izmantot, vai arī pārveidot pašus datus. Piemērs varētu būt, BLOB datu tips Oracle datu bāzē un SQL Server datu bāzē. Oracle atļauj izmantot vienas tabulas ietvaros vairākus BLOB-veidīgo atribūtu, bet SQL Server tikai vienu uz vienu tabulu. Tātad ja tabulā Oracle datu bāzē bija vairāki BLOBi, tad migrējot uz SQL Server datu bāzi, vajadzēs pārtaisīt tabulu, sadalot to vairākās.

b. Tabulu un kolonu īpašības

   NULL vērtības, primāras un ārējas atslēgas, unikālie (unique) un pārbaudes ierobežojumi (check constraint), lauku komentāri, secības (sequence). Ja kaut kas no šim īpašībām netiek uzturēts jaunajā datu bāzē, tad būs jātaisa papildus funkcionalitāte, lai tas arī tiktu atbalstīts. T.i., piemēram, ja jaunajā datu bāzē nevar uztaisīt unikālo ierobežojumu, tad jāraksta trigeris uz datu ielikšanu (insert) un modificēšanu (update) tabulā, kas pārbaudīs vai tāda vērtība jau nav saglabāta.

c. Maksimālā garuma ierobežojumi

Katrai datu bāzei maksimāli pieļaujamie garumi var atšķirties, t.i. datu bāzes identifikatoriem, tabulas nosaukumiem, mainīgiem, u.t.t. Ja vecajā datu bāzē bija, piemēram, tabulas nosaukums garāks nekā tas ir pieļauts jaunajā datu bāzē, tad samazinot to, ir obligāti jāizmaina šo pašu „nekorekto" nosaukumu visas vietās, kur tas ir izmantots (SQL vaicājumos, skatījumos, trigeros, procedūrās, u.t.t.).

d. Datu bāzes rezervētie vārdi

Bieži migrējot no vienas datu bāzes uz otru gadās situācijas, kad vienā datu bāzē kaut kāds vārds ir mainīgais (vai tabulas nosaukums), bet otrā tas ir rezervētais vārds, t.i. vienkārši pārkopēt tādu mainīgu nedrīkst. Piemēram, ja vecajā datu bāzē mums bija tabulas lauka nosaukums „Ltrim" un patreiz mēs gribam migrēt uz Oracle, tad parastais SQL vaicājums ar šo lauku izradīsies kļūdains, jo Oracle datu bāzē šī ir iebūvēta funkcija.

e. Lēna datu importēšana

Ja datu bāze ir pārāk liela (piemēram, ir vairākas tabulās, kuras tiek papildinātas katru dienu ar ap 1 miljonu ierakstu), tad, piemēram, ieimportēt datus tabulās (pat ja tās ir patreiz bez indeksiem un ar pilnīgi tādu pašu tabulas struktūru kā iepriekšējā datu bāze) nebūs ātri. No šis problēmas varētu izvairīties, ja ir speciāli rīki, kuri spēj pietiekoši ātri importēt datus tabulās ar lielu datu apjomu.

f. Skatījumu pārnešana

Skatījums principā ir tas pats SELECT teikums, izveidots un saglabāts SQL vaicājumu ērtībai un uzskatāmībai. Taču katrā datu bāzē ir savi vaicājumu „dialekti" (piemēram, Outer Join sintakse atšķiras dažādās datu bāzēs), vaicājuma struktūras, funkcijas un izņēmumi. Kaut gan šī

nav tā lielāka problēma, jo ir SQL valodas standarti (SQL-86 vai SQL-87, SQL-89, SQL-92 vai SQL2, SQL:1999 vai SQL3, SQL:2003) [7] un tas nozīme, ja datu bāzes izmanto tos pašus standartus, tad būtiskas SQL vaicājumu atšķirības nebūs.

g. Trigeru, procedūru un funkciju konvertācija
   Šī ir vissvarīgākā un nopietnāka problēma, kas rodas migrācijas procesā, jo katras datu bāzes procedūru valodai ir sava pilnīgi atšķirīga sintakse. Piemēram, Oracle datu bāzei procedūru valoda ir PL/SQL, bet ADABAS D ir SQL-PL, un šis valodas nopietni atšķiras viena no otras, tas ir jāņem vērā. Vispār jebkurai datu bāzei ir sava procedūru valoda, ar savam īpatnībām, iebūvētam funkcijām un semantiku [2]. Un parasti šīajās vietās arī ir vairākums no projekta operācijām ar datu bāzi.

2.5 Migrācijas problēmu risinājumi

Datu bāzes migrācija nav nekas jauns un tā jau pastāv ļoti ilgu laiku. Tieši tāpēc lai atvieglot dzīvi ir izveidoti vairāki rīki, kas palīdz vai pat pilnībā veic datu bāzes migrāciju. Tālāk tiek pārskaitīti populārākie datu bāzes migrācijas rīki.

- *Oracle Migration WorkBrench*. Tas ir rīks, kas padara migrācijas procesu vieglāku. Tas migrē datu bāzes shēmu, ieskaitot trigerus un iebūvētas procedūras esošā vidē. Šīs rīks atbalsta migrāciju no Oracle (jaunākas versijas), Sybase, DB2, UDB, Access, SQL Server, MySQL uz Oracle datu bāzi.

- *Oracle Database Migration Verifier*. Tas ir rīks, ar kuru palīdzību kontrolē Oracle migrācijas. Tas analizē primāro datu bāzi un salīdzina to ar Oracle datu bāzi, lai apstiprinātu struktūras un datu integritāti uz migrēto Oracle datu bāzi. Šīs rīks ģenerē detalizētu atskaiti ar apkopotiem testu

rezultātiem pēc shēmas un datu salīdzinošām pārbaudēm. Šis rīks atbalsta Oracle un Sybase, SQL Server datu bāzes salīdzināšanu.

- *AdventNet SwisSQL Database Migration Suite*:
  a. *Stored Procedure Migration rīki* pilnība automatizē iebūvēto procedūru migrāciju no SQL Server Transact-SQL, Sybase Transact-SQL un IBM DB2 SQL uz Oracle PL/SQL.
  b. *SQL Migration rīks* pilnība migrē SQL vaicājumus no MS SQL Server, IBM DB2 UDB, Sybase ASE, MySQL, Informix un PostgreSQL uz Oracle SQL vaicājumiem.
  c. *Data Migration rīks* pilnība migrē primāras datu bāzes shēmu un datus uz Oracle datu bāzi. Šis rīks atšķirībā no citiem šīs firmas produktiem spēj migrēt ne tikai uz Oracle, skatīt Tabula 1 [9].

Primāras datu bāzes	Mērķa datu bāzes
Oracle 8i,9i un 10g	Oracle 9i un 10g
IBM DB2 UDB for Linux, Unix, and Windows 8.1, 8.2	IBM DB2 UDB for Linux, Unix, and Windows 8.1, 8.2
MS SQL Server 2000	MS SQL Server 2000
Sybase ASE 12.x, 11	Sybase ASE 12.x un 11
MySQL 4.0.x un 3.23.x	MySQL 4.0.x, 3.23.x, un 5.0 Beta
MySQL MaxDB (SAP DB) 7.5.00	

Tabula 1 – AdventNet SwisSQL Data Migration rīka iespējamas datu bāzes

- *SQLWays*, kuru piedāvā Ispirer Systems Ltd kompānija. Šīs rīks ir pilnīgs migrācijas risinājums, tas automatizē migrācijas procesu konvertējot datus, datu bāzes objektus, ierobežojumus (constraints) un biznesa loģiku. Šo rīku var principā pielāgot unikālajam prasībām, lai, piemēram, konvertēt projektus ar vairākām datu bāzēm. Šis rīks atbalsta vairākas datu bāzes un izmanto tehnoloģijas, kas nezaudējot ātrdarbību spēj strādāt ar lielo datu apjomiem. Datu bāzes, kuras atbalsta SQLWays ir pārskaitītas Tabula 2 [10].

Primāras datu bāzes	Mērķa datu bāzes
IBM DB2 for Linux, Unix and Windows 8.2, 8.1, 7.2, 7.1, 7.0, 6.1 un jaunākas Oracle 10g, 9i, 8i, 8.0.x un 7.x Microsoft SQL Server 2005, 2000, 7.0 un 6.5 Sybase Adaptive Server Enterprise (ASE) 12.5.1, 12.5, 12.0, 11.x un jaunākas Sybase Adaptive Server Anywhere (ASA), Sybase SQL Anywhere 9.0, 8.0.x, 7.0.x, 6.0.x, 5.5 un 5.0 Sybase IQ 12.x Informix Dynamic Server (IDS) 10, 9.4, 9.3, 9.2, 9.1, 7.3, 7.2, 7.1 un jaunākas Informix Standard Engine (SE) 7.x, 5.x, 4.x un jaunākas MySQL 5.x, 4.x, 3.23 un jaunākas PostgreSQL 8.x, 7.x un 6.x Progress 9.x un 8.x SAP DB 7.4, 7.3 un jaunākas Pervasive.SQL v8, 2000 un 7 Microsoft Access 2003, 2000, 97, 95 un 2.0 Interbase, Firebird 7.1, 6.x, 5.x un 4.x Lotus Notes 6, 5 un jaunākas dBase, FoxPro, Excel, Paradox, Gupta SQLBase, Clipper un citas, kas ir ODBC-atbilstīgas datu bāzēs	IBM DB2 for Linux, Unix and Windows 8.2, 8.1, 7.2, 7.1, 7.0, 6.1 un jaunākas Oracle 10g, 9i, 8i un 8.0.x Microsoft SQL Server 2005, 2000, 7.0 un 6.5 MySQL 5.x, 4.x un 3.23 PostgreSQL 8.0 un 7.x Sybase Adaptive Server Enterprise 12.5.1, 12.5, 12.0, 11.x Sybase IQ 12.x Informix Dynamic Server (IDS) 10, 9.4, 9.3, 9.2, 9.1, 7.3, 7.2, 7.1 un jaunākas Pervasive.SQL v8

Tabula 2 – SQLWays rīka iespējamas datu bāzes

Protams, ir daudz vairāk rīku, kurus var izmantot datu bāzes migrācijai, bet mans mērķis nebija tos visus šeit pārskaitīt.

3 Kāpēc Oracle un ADABAS D...

Oracle mūsdienas ir ļoti izplatīta un attīstīta datu bāzes pārvaldības sistēma. Protams, Oracle ir arī konkurenti, vismaz tas DBPS, kuras Oracle uzskata par saviem konkurentiem:

- CA with Ingres
- IBM with DB/2
- Informix with the Informix DB
- Microsoft with Access and SQL Server
- Software AG with ADABAS D
- Sybase with their Sybase System
- PostgreSQL (free open source database) [**8**]

Es pati esmu Oracle programmētājs un visu laiku bija interesanti cik ļoti datu bāzes atšķiras viena no otras un kādā veida var pāriet, piemēram, no Oracle uz kādu citu. Tātad pieredzes dēļ tika izvelēts Oracle DBPS, un tā kā mana bakalaura darba vadītājs pārzin ADABAS D DBPS un Oracle uzskata to kaut cik konkurēt spējīgu sev, tad to es paņemu kā otru datu bāzi.

3.1 Oracle

3.1.1 Oracle vēsture

Oracle korporācija bija dibināta 1977. gadā, Redwoodā, Kalifornijā. Korporācijas dibinātāji pirmie ieviesa relāciju datu bāzes vadības sistēmu. Tā bija bāzēta uz IBM System/R modeļa un pielietoja datu bāzes vadības sistēmas IBM's Structured Query Language (SQL) tehnoloģiju. Pirmā Oracle versija iznāca 1979 gadā un bija nosaukta Oracle v2 (Oracle v1 nekad nebija – marketinga triks). Patreiz pēdēja versija ir Oracle 10g.

Šodien Oracle datu bāzes vadības sistēmas ir uzturētas vairāk nekā 80 dažādās strādājošās vidēs, piemēram, IBM lieldatoros, minidatoros, UNIX – bāzētas minidatoros, Windows NT un citās operētājsistēmās un platformās. Oracle

nodarbina vairāk nekā 42,000 profesionāļus 93 valstīs visā pasaulē. Skaidrs, ka tas šodien ir lielākais relāciju datu bāzes pārvaldības sistēmas pārdevējs.

Vārda **Oracle** skaidrojumu vārdnīcās var atrast, kā:
"Prophecy or prediction; answer to a question believed to come from the gods; a statement believed to be infallible and authoritative; a shrine at which these answers are given"

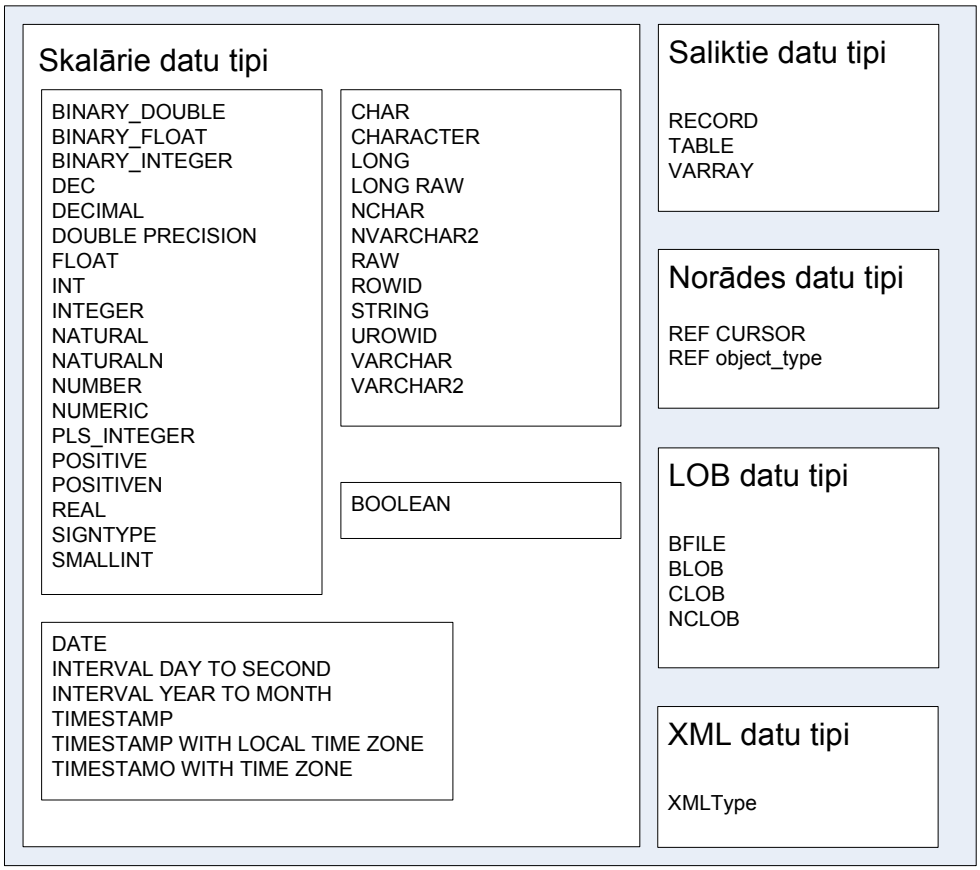
Sakumā tas nosaukums bija izmantots datu bāzes kodolam, vēlāk arī kompānijai. Larry Ellison un Bob Miner strādāja konsultējošā CIA (Central Intelligence Agency in USA) projektā, kur CIA gribēja ieviest un izmantot jaunu IBM programmēšanas valodu SQL. Nosaukums projektam bija Oracle, tāpēc, ka CIA redzēja savu sistēmu kā atbilde uz visiem jautājumiem. Projekts drīzumā 'nomira', bet Larry Ellison un Bob Miner redzēja izdevību paņemt un turpināt to, kas bija iesakts. Tāda veidā viņi sāka izmantot Oracle nosaukumu savam jaunajam relāciju datu bāzes vadības sistēmas kodolam [**8**].

3.1.2 SQL un PL/SQL valodas

Oracle PL/SQL valoda ir labi aprakstīta vairākās grāmatās, es izmantoju [11, 12, 13].

Datu tipi

Oracle ir pārāk daudz datu tipu, tāpēc tos vieglāk ir attēlot tabulas veidā (Zīm. 3).



Zīm. 3 – Oracle datu tipi

Mainīgie

Oracle mainīgus var definēt pakotnes definīcijā kā globālus mainīgus vai arī pēc atslēgvārda DECLARE PL/SQL blokā. Sintakse mainīga deklarācijai ir

```
var_name datatype [CONSTANT] [NOT NULL] [:= | DEFAULT initial_value];
```

Mainīgam var uzreiz piešķirt noklusēto vērtību ar piešķirošu operatoru (:=) vai DEFAULT atslēgvārdu.

Var definēt mainīgu ar cita mainīga vai tabulas kolonnas datu tipu izmantojot atribūtu %TYPE. Un lai dabūt mainīgus no tabulas rindas ar līdzīgiem datu tipiem izmanto atribūtu %ROWTYPE.

Operatori

- Aritmētiskie operatori: +, -, *, /, ** (tikai PL/SQL izmantošanai)
- Konkatenācijas operators: ||
- Grupu (set) operatori: UNION, UNION ALL, MINUS, INTERSECT

- Oracle var definēt arī savus operatorus

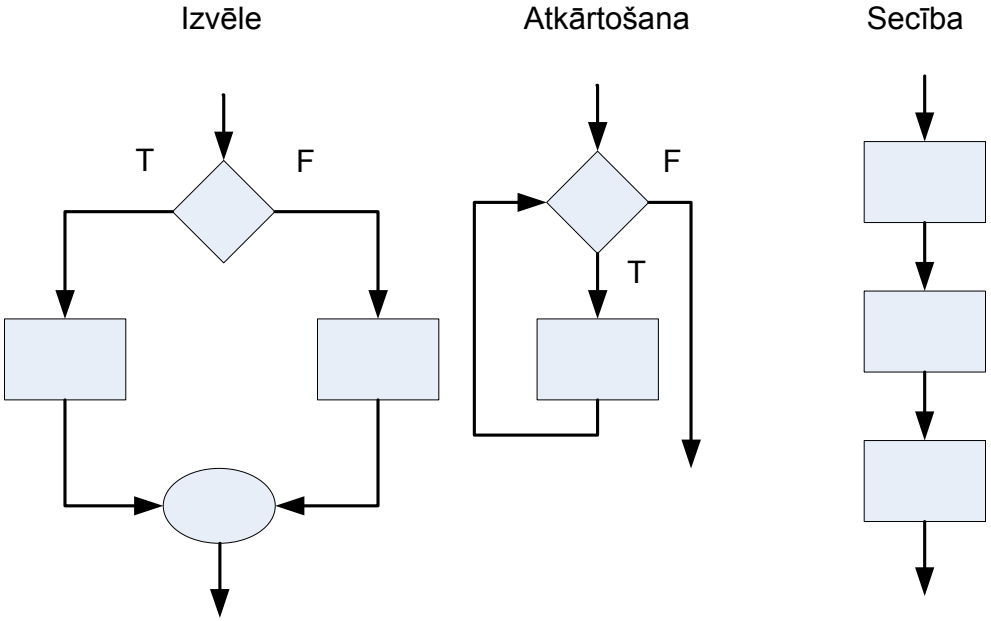
```
CREATE OR REPLACE OPERATOR <operator_name> BINDING (data_type_in)
RETURN <data_type_out> USING <function_name>;
```

Nosacījumi (Conditions)

- Salīdzinājuma nosacījumi: =, !=, <>, ^=, ¬=, <, >, <=, >=, ANY, SOME, ALL
- Loģiskie nosacījumi: NOT, AND, OR
- Piederības nosacījumi: IN, NOT IN
- Diapazona nosacījumi: [NOT] BETWEEN x AND y
- NULL nosacījumi: IS [NOT] NULL
- EQUALS_PATH nosacījums
- EXISTS nosacījums
- LIKE nosacījums: x [NOT] LIKE y [ESCAPE ‘z’]
- IS OF nosacījums
- UNDER_PATH nosacījums

PL/SQL kontroles struktūra

Oracle izmanto bāzes kontroles struktūras (Zīm. 4).



Zīm. 4 – Kontroles struktūras

Pie izvēles struktūras pieder:

- IF-THEN izteikums;
- IF-THEN-ELSE izteikums;
- IF-THEN-ELSIF izteikums;
- CASE izteikums.

Pie atkārtošanas struktūras pieder:

- LOOP izteikums;
- WHILE-LOOP izteikums;
- FOR-LOOP izteikums

Pie secību struktūras pieder:

- GOTO izteikums;
- NULL izteikums.

Oracle SQL

Oracle 9i izmanto SQL ANSI/ISO standartu – SQL:1999 (vai tā saucamo SQL:3).

PL/SQL moduļi

Oracle PL/SQL atbalsta sekojošus moduļu veidus:

- procedūras;
- funkcijas;
- pakotnes;
- trigeri;
- objektu tipi.

Procedūras

Procedūras ir programmas, kas pilda vienu vai vairākus izteikumus, mēs varam padot parametrus un caur parametriem saņemt rezultātus. Procedūru definēšanas shēma:

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter [,parameter]) ]
IS | AS
  declaration_section
BEGIN
  executable_section
[EXCEPTION
```

```
  exception_section]
END [name];
```

Izsaukt procedūras var sekojošā veidā:

```
BEGIN
  EXECUTE proc_name(p_par1 => v_par1, p_par2 => v_par2);
END;
```

Funkcijas

Funkcijas ir programmas, kas atgriež kādu vērtību. Oracle ir divi funkciju veidi sistēmā definētas un lietotāju definētas.

Simbolu/Virknes funkcijas:			
Ascii	Convert	Lower	Soundex
AsciiStr	Decompose	Lpad	Substr
Chr	Dump	Ltrim	Translate
Compose	Initcap	Replace	Trim
Concat	Instr	Rpad	Upper
Concat with ||	Length	Rtrim	VSize
Pārveidošanas funkcijas:			
Bin_To_Num	NumToDSInterval	To_DSInterval	To_Single_Byte
Cast	NumToYMInterval	To_Lob	To_Timestamp
CharToRowid	To_Char	To_Multi_Byte	To_Timestamp_Tz
From_Tz	To_Clob	To_NClob	To_YMInterval
HexToRaw	To_Date	To_Number	
Vispārējas funkcijas:			
BFilename	Decode	NVL2	UserEnv
Cardinality	Group_ID	Sys_Context	
Case Statement	NANVL	Uid	
Coalesce	NVL	User	
Matemātiskas funkcijas:			
Abs	Covar_pop	Max	Sqrt
Acos	Covar_samp	Median	StdDev
Asin	Count	Min	Sum
Atan	Cume_Dist	Mod	Tan
Atan2	Dense_Rank	Power	Tanh
Avg	Exp	Rank	Trunc (numbers)
Bin_To_Num	Extract	Remainder	Trunc (dates)
BitAnd	Floor	Round (numbers)	Var_pop
Ceil	Greatest	Round (dates)	Var_samp
Corr	Least	Sign	Variance
Cos	Ln	Sin	
Cosh	Log	Sinh	
Datuma funkcijas:			

Add_Months	Last_Day	Round	To_Date
Current_Date	LocalTimestamp	SessionTimeZone	Trunc
Current_Timestamp	Months_Between	Sysdate	Tz_Offset
DbTimeZone	New_Time	SysTimestamp	
From_Tz	Next_Day	To_Char	
Kļūdas funkcijas:			
SQLCODE			
SQLERRM			

Tabula 3 – Oracle iebūvētas funkcijas

Lietotāju definētas funkcijas principā ir tas pašas procedūras ar vienīgo atšķirību – funkcijai ir jāatgriež vērtība.

Funkcijas definēšanas shēma:

```
CREATE [OR REPLACE] FUNCTION name
[ (parameter [,parameter]) ]
RETURN return_datatype
IS | AS
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

Funkcijas var izsaukt sekojošos veidos:

- ar piešķiršanas operatora palīdzību

```
BEGIN
  v_rez := funct_name(p_par1 => v_par1, p_par2 => v_par2);
END;
```

- parametram vērtība pēc noklusējuma

```
DECLARE
  v_variable NUMBER DEFAULT funct_name(p_par1 => v_par1);
BEGIN
```

- kā loģiska izteiksme

```
IF funct_name(p_par1 => v_par1) THEN
```

- SQL vaicājumā

```
SELECT *
FROM table_name
WHERE col_name = funct_name(p_par1 => v_par1);
```

- Kā arguments cita programmā parametru sarakstā

```
proc_name(v_param1, today_result(SYSDATE));
```

Pakotnes

Pakotnes ir procedūru, funkciju un datu struktūru konteineri. Pakotnes sastāv no divām daļām pakotnes specifikācija (galva) un pakotnes ķermenis.

Pakotnes specifikācijas definēšanas shēma:

```
CREATE [OR REPLACE] PACKAGE package_name
IS | AS
[definitions of public TYPEs
,declarations of public variables, types and objects
,declarations of exceptions
,pragmas
,declarations of cursors, procedures and functions
,headers of procedures and functions]
END [package_name];
```

Pakotnes ķermeņa definēšanas shēma:

```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS | AS
[definitions of private TYPEs
,declarations of private variables, types AND objects
,full definitions of cursors
,full definitions of procedures and functions]
[BEGIN
executable_statements
[EXCEPTION
exception_handlers ] ]
END [package_name];
```

Pakotnes specifikācijā noradītie objekti var būt izmantoti ārpus konkrētas pakotnes ietvariem (publiskie objekti), toties ja objekti ir definēti pakotnes ķermenī, tad tos var izmantot tikai konkrētā pakotnē (privātie objekti).

Trigeri

Trigeris ir programma, kas izpildās sakarā ar datu bāzes izmaiņām. Trigeri var „reaģēt" uz sekojošam datu bāzes darbībām:

- INSERT
- UPDATE
- DELETE

Trigeru definēšanas shēma:

```
BEFORE | AFTER | INSTEAD OF trigger_event
ON
[ NESTED TABLE nested_table_column OF view ]
| table_or_view_reference | DATABASE
trigger_body;
```

Trigerus var ieslēgt un atslēgt ar sekojošo skriptu:

```
ALTER TRIGGER trigger_name ENABLE | DISABLE;
```

Objektu tipi

Objektu tips ir salikts datu tips, kas reprezentē datu struktūru, procedūras un funkcijas datu manipulēšanai. Objektu tips apvieno sevī atribūtus (datu struktūras) un metodes (funkcijas un procedūras). Objektu tips līdzīgi pakotnei sastāv no specifikācijas un ķermeņa.

Objektu tipa specifikācijas definēšanas shēma:

```
CREATE [OR REPLACE] TYPE obj_type_name
AS OBJECT (
attribute_name datatype,...,
[MEMBER | STATIC PROCEDURE | FUNCTION
program_spec],
[ORDER | MAP MEMBER FUNCTION
comparison_function_spec],
[PRAGMA RESTRICT_REFERENCES(program_name, purities)]
);
```

Objektu tipa ķermeņa definēšanas shēma:

```
CREATE [OR REPLACE] TYPE BODY obj_type_name
AS OBJECT (
[MEMBER | STATIC PROCEDURE | FUNCTION
program_body;]
[ORDER | MAP MEMBER FUNCTION
comparison_function_body;]
);
```

Kļūdu apstrāde

PL/SQL programmās kļūdas saucas par izņēmumiem. Izņēmumi var būt sistēmas un lietotāju definētie izņēmumi. Dažiem sistēmas definētiem izņēmumiem ir nodefinēti nosaukumi (piemēram, ZERO_DEVIDE). Lai izsaukt lietotāju definētos izņēmumus ir jāizmanto atslēgvārdu RAISE. Izsauktos izņēmumus apstrādā Oracle definētie izņēmumu apdarinātāji (Tabula 4). Lai reģistrēt savu izņēmumu ir jāizmanto BEGIN-END bloku, kurā ja kaut kas neatbilst prasībām mēs izsaucam izņēmumu, no kura seko tālākas darbības.

Izņēmums	Oracle kļūda	SQLCODE vērtība
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531

CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Tabula 4 – Oracle definētie izņēmumi

Piemērs programmas atkļūdošanai:

```
DECLARE
  v_error EXCEPTION;
  v_tmp NUMBER;
BEGIN

  BEGIN
    SELECT COUNT(1)
    INTO v_tmp
    FROM customers
    WHERE name LIKE 'Alex%' AND sex = 'F';

    IF v_tmp = 1 THEN
      UPDATE customers
      SET name = 'Aleksejs', sex = 'M'
      WHERE name LIKE 'Alex%' AND sex = 'F';
    ELSE
      RAISE v_error;
  EXCEPTION
    WHEN v_error THEN
      DELETE * FROM customers WHERE name LIKE 'Alex%' AND sex = 'F';
    WHEN OTHERS THEN
      dbms_output.put_line('Neparedzeta kluda: '||SQLERRM);
      ROLLBACK;
  END;
```

```
  COMMIT;
END;
```

Oracle rezervētie vārdi

ALL*	EXECUTE	NOCOPY	SHARE*
ALTER*	EXISTS*	NOT*	SMALLINT*
AND*	EXIT	NOWAIT*	SPACE
ANY*	EXTENDS	NULL*	SQL
ARRAY	EXTRACT	NULLIF	SQLCODE
AS*	FALSE	NUMBER*	SQLERRM
ASC*	FETCH	NUMBER_BASE	START*
AT	FLOAT*	OCIROWID	STDDEV
AUTHID	FOR*	OF*	SUBTYPE
AVG	FORALL	ON*	SUCCESSFUL*
BEGIN	FROM*	OPAQUE	SUM
BETWEEN*	FUNCTION	OPEN	SYNONYM*
BINARY_INTEGER	GOTO	OPERATOR	SYSDATE*
BODY	GROUP*	OPTION*	TABLE*
BOOLEAN	HAVING*	OR*	THEN*
BULK	HEAP	ORDER*	TIME
BY*	HOUR	ORGANIZATION	TIMESTAMP
CASE	IF	OTHERS	TIMEZONE_REGION
CHAR*	IMMEDIATE*	OUT	TIMEZONE_ABBR
CHAR_BASE	IN*	PACKAGE	TIMEZONE_MINUTE
CHECK*	INDEX*	PARTITION	TIMEZONE_HOUR
CLOSE	INDICATOR	PCTFREE*	TO*
CLUSTER*	INSERT*	PLS_INTEGER	TRIGGER*
COALESCE	INTEGER*	POSITIVE	TRUE
COLLECT	INTERFACE	POSITIVEN	TYPE
COMMENT*	INTERSECT*	PRAGMA	UID*
COMMIT	INTERVAL	PRIOR*	UNION*
COMPRESS*	INTO*	PRIVATE	UNIQUE*
CONNECT*	IS*	PROCEDURE	UPDATE*
CONSTANT	ISOLATION	PUBLIC*	USE
CREATE*	JAVA	RAISE	USER*
CURRENT*	LEVEL*	RANGE	VALIDATE*
CURRVAL	LIKE*	RAW*	VALUES*
CURSOR	LIMITED	REAL	VARCHAR*
DATE*	LOCK*	RECORD	VARCHAR2*
DAY	LONG*	REF	VARIANCE
DECLARE	LOOP	RELEASE	VIEW*
DECIMAL*	MAX	RETURN	WHEN
DEFAULT*	MIN	REVERSE	WHENEVER*
DELETE*	MINUS*	ROLLBACK	WHERE*
DESC*	MINUTE	ROW*	WHILE
DISTINCT*	MLSLABEL*	ROWID*	WITH*
DO	MOD	ROWNUM*	WORK
DROP*	MODE*	ROWTYPE	WRITE
ELSE*	MONTH	SAVEPOINT	YEAR
ELSIF	NATURAL	SECOND	ZONE
END	NATURALN	SELECT*	
EXCEPTION	NEW	SEPARATE	
EXCLUSIVE*	NEXTVAL	SET*	

Tabula 5 – Oracle PL/SQL rezervētie vārdi (vārdi ar * ir rezervēti arī SQL)

3.1.3 Piedāvātie rīki

Principā Oracle peidādvā plašu standartā paketi ar iebūvētiem rīkiem, taču ir arī daudz tādu rīku, kas ir ne Oracle rīki. Tē ir pārskaitīti ti rīku, kas mums būs nepieciešami migrācijai:

- Administrēšanas rīks Enterprise Manager Console – galvenais lokālais administratora rīks. Ar šo rīku var administrēt sekojošo:
  - datu bāzes palaišanu (startup) un izslēgšanu (shutdown), kā arī tās inicializāciju;
  - shēmas, ieskaitot tabulas, indeksus, un parējos shēmas objektus;
  - drošību, ieskaitot lietotājus, lomas un privilēģijas;
  - krātuvi, ieskaitot tabulvietas (tablespaces), datu failus un rollback segmentus;
  - datu noliktavas, ieskaitot vispārīgu vadību un OLAP vadību;
  - citas datu bāzes īpatnības (tas ir atkarīgs no tā, kādi produkti ir uzinstalēti kopā ar datu bāzi).
- SQL Plus – rīks ir paredzēts darbam ar datu bāzi (SQL vaicājumu pildīšana, iebūvēto procedūru izsaukumi/modifikācijas, datu bāzes parametru konfigurācija, u.t.t.). Tas ir uztaisīts pēc komandrindas principā.
- SQL Loader – rīks ir paredzēts liela apjoma datu ielādei datu bāzē.

3.2 ADABAS D

3.2.1 ADABAS D vēsture

ADABAS D tika palaists 1975. gadā. To izstrādāja Berlīnes Tehniskā Universitāte (saucās VDN Verteilte Datenbank Netz). Taču 1978. gadā to nopirka kompānija Nixdorf Computer AG un 1987 šo datu bāzes pārvaldības sistēmu pārsauca par DB4. Un beidzot 1992. gadā DB4 tika pārpirkts ar kompāniju Software AG. Un tā kā šai kompānijai jau sen tika izstrādāta datu bāzes pārvaldības sistēma ADABAS C lieldatoriem (kuru parasti sauc par ADABAS), tad tika nolemts šo DBPS nosaukt ADABAS D. Taču starp ADABAS C un ADABAS D nav nekā kopīga, vienīgi nosaukumi ir līdzīgi.

Dažādos laikos (pēc 1992) patstāvīgu dzīvi sāk no šī paša koda ar citu nosaukumu: SAP DB un MAXDB.

ADABAS D tika reklamēts kā datu bāzes pārvaldības sistēma spējīga aizstāt tādas DBPS kā Oracle, SQL Server, DB2 un citas. ADABAS D spēj strādāt zem vairākām platformām: Windows XP, Windows 2000, Windows Server 2003, Unix, Linux, AIX, HP-UX 11i un Sun Solaris. Patreiz pēdēja versija ir ADABAS D 13.01.

3.2.2 SQL un SQL-PL valodas

Adabas D SQL-PL valoda ir labi aprakstīta vairākās grāmatās, es izmantoju [14, 15].

Datu tipi

Adabas D atbalsta sekojošos datu tipus:

- CHAR
- VARCHAR
- BOOLEAN
- DATE
- TIME

- TIMESTAMP
- FIXED
- SERIAL/AUTOINCREMENT
- FLOAT
- LONG
- INTEGER
- SMALLINT
- DECIMAL
- FLOAT
- DOUBLE
- PRECISION
- REAL
- LONG VARCHAR
- BYTE

Mainīgie

SQL-PL mainīgie var būt skalārie un vektora mainīgie. Definējot mainīgos, tā datu tips nav jānorāda. Mainīgie var saturēt vai nu skaitlisku, vai simbolisku vērtību, vai arī būt ar nedefinētu (NULL) vērtību.

Pēc darbības apgabala SQL-PL mainīgos sadala sekojoši:

- Globālie mainīgi;
- Lokālie dinamiskie mainīgi (apzīme ar @);
- Lokālie statiskie mainīgi (apzīme ar @@).

Globālā mainīgā vērtība ir pieejama jebkurā programmas procedūrā. Lokālā dinamiskā mainīgā un lokālā statiskā mainīgā vērtība ir pieejama tikai tajā programmas modulī, kur mainīgais definēts (mainīgus definē ar atslēgvārdu VAR). Lokālā statiskā mainīgā vērtība atšķirībā no lokālā dinamiskā mainīgā vērtības saglabājas līdz nākošajam moduļa izsaukumam.

Piemērs:

`x(@y) := @@z;`	
`x`	`Ir globāls vektors`
`@y`	`Ir lokālais dinamiskais skalārais`

@@z	Ir lokāls statisks skalārais

Operatori

- Aritmētiskie operatori: +, -, *, /, MOD, DIV
- Konkatenācijas operators: &

Nosacījumi (Conditions)

- Salīdzinājuma nosacījumi: =, <>, <, >, <=, >=,
- Loģiskie nosacījumi: NOT, AND, OR
- Piederības nosacījumi: IN, NOT IN
- Diapazona nosacījumi: [NOT] BETWEEN x AND y
- NULL nosacījumi: IS [NOT] NULL
- SOUNDS nosacījums
- LIKE nosacījums: x [NOT] LIKE y [ESCAPE ‘z’]
- IS nosacījumi: IS [TRUE | FALSE]

SQL-PL kontroles struktūras

Adabas D izmanto bāzes kontroles struktūras (Zīm. 4).

Pie izvēles struktūras pieder:

- IF-THEN izteikums;
- IF-THEN-ELSE izteikums;
- CASE izteikums.

Pie atkārtošanas struktūras pieder:

- WHILE-DO izteikums;
- FOR-TO izteikums;
- FOR-DOWNTO izteikums;
- REPEAT-UNTIL izteikums.

Pie secību struktūras pieder:

- SKIP izteikums.

Adabas D SQL

Adabas D 13.01 izmanto SQL ANSI/ISO standartu – SQL-92 (vai tā saucamo SQL:2).

SQL-PL moduļi

Adabas D atbalsta sekojošus SQL-PL moduļus:

- DB procedūras
- trigeri
- DB funkcijas
- procedūras
- funkcijas
- formas
- HELP formas
- izvēlnes

Taču es savā bakalaura darbā neizskatīšu tādus moduļus kā formas, HELP formas un izvēlnes migrāciju. Kaut arī Oracle arī atbalsta šos moduļus, tas nav tik vienkārši migrēt vienu iebūvēto interfeisu uz otru, jo tas pirmkārt ir atšķirīgas valodas un otrkārt vizuāli un tehniski Oracle un Adabas D formas principiāli atšķirsies.

DB procedūras

DB procedūras ir SQL-PL programmas, kas tiek izsauktas no lietojumprogrammas kā viens SQL izteikums. DB procedūrās tiek atļauts izmantot SQL vaicājumus. No DB procedūrām var izsaukt procedūras un funkcijas vai tālākas DB procedūras. Izmantojot SQL vaicājumus ir iespēja izsaukt trigerus un DB funkcijas. DB procedūrās nav atļauts veikt transakcijas. Kā arī no DB procedūrām ir iespējams izsaukt OS komandas vai lietojumprogrammatūras programmas (C, COBOL).

DB procedūras definēšanas shēma:

```
DBPROC prog_name.mod_name
[PARMS (parameter [,parameter])]
[OPTIONS (module_option [,module_option])]
[declaration_section]
```

```
executable_section
```

DB procedūru izsaukums no SQL-PL:

```
CALL DBPROC [[owner.]progname.]dbproc_name [PARMS (param1, ...)]
[WITH COMMIT]
| SQL ( DBPROC[EDURE] [[owner.]progname.]dbproc_name [(host_var,
....)] [WITH COMMIT])
```

DB procedūru izsaukums no prekompilatora (precompiler):

```
EXEC SQL (DBPROCEDURE owner.appl.module (:var1, :var2, ...))
```

Trigeri

Trigeris ir programma, kas izpildās sakarā ar datu bāzes izmaiņām. Trigeri var „reaģēt" uz sekojošam datu bāzes darbībām:

- INSERT
- UPDATE
- DELETE

No trigeriem ir atļauts izsaukt funkcijas, procedūras un DB procedūras. Trigeru parametriem ir jāsakrīt ar tabulas kolonu nosaukumiem un datu tipiem.

Trigeru definēšanas shēma:

```
TRIGGER prog_name.mod_name
[PARMS (parameter [,parameter])]
[OPTIONS (module_option [,module_option])]
[declaration_section]
executable_section
```

DB funkcijas

Adabas D ir savas iebūvētas funkcijas:

Simbolu/Virknes funkcijas:			
SUBSTR	LTRIM	RPAD	ASCII
|| (concatenation)	RTRIM	MAPCHAR	EBCDIC
& (concatenation)	EXPAND	INITCAP	SOUNDEX
LFILL	UPPER	REPLACE	
RFILL	LOWER	TRANSLATE	
TRIM	LPAD	ALPHA	
Pārveidošanas funkcijas:			
NUM	CHR	HEX	CHAR
Vispārējas funkcijas:			
VALUE	GREATEST	DECODE	LEAST
Matemātiskas funkcijas:			

TRUNC	POWER	COT	ATAN2
ROUND	SQRT	COSH	RADIANS
FIXED	LENGTH	SINH	DEGREES
CEIL	INDEX	TANH	EXP
FLOOR	COS	ACOS	LN
SIGN	SIN	ASIN	LOG
ABS	TAN	ATAN	PI
DIV	MAX	AVG	
MOD	MIN	STDDEV	
COUNT	SUM	VARIANCE	
Datuma funkcijas:			
ADDDATE	DAYOFMONTH	YEAR	DAYNAME
SUBDATE	WEEKOFYEAR	MONTH	MONTHNAME
DATEDIFF	DAYOFYEAR	DAY	
DAYOFWEEK	MAKEDATE	TIMESTAMP	
Laika funkcijas:			
ADDTIME	MAKETIME	SECOND	SUBTIME
TIMEDIFF	HOUR	TIME	MINUTE
MICROSECOND			

Bet, ja gadījumā nav vajadzīgas iebūvētas funkcijas, tad tiek izmantotas DB funkcijas. Šīs funkcijas pēc tam var izmantot SQL vaicājumos. DB funkcijām var padot vairākus parametrus, bet atgriezt tikai vienu. DB funkcijās nedrīkst izmantot nekādus SQL vaicājumus. DB funkcijas nosaukumam ir jābūt unikālam visa datu bāzes līmenī.

DB funkcijas definēšanas shēma:

```
DBFUNC prog_name.mod_name
[PARMS] (parameter [,parameter]) : data_type
[OPTIONS (LIB prog_name)]
[declaration_section]
executable_section
```

Kā piemērs DB funkciju izsaukumam ir:

```
SQL (SELECT name, fupper(name) FROM customers);
```

Procedūras

Procedūras ir programmas, kas pilda vienu vai vairākus izteikumus, mēs varam padot parametrus un caur parametriem saņemt rezultātus. No procedūrām var izsaukt citas procedūras, funkcijas un DB procedūras. Kā arī caur SQL vaicājumiem var izsaukt trigerus vai DB funkcijas.

```
PROC prog_name.mod_name
[[PARMS] (parameter [,parameter])]
[OPTIONS (module_option [,module_option])]
```

```
[declaration_section]
executable_section
```

Procedūru izsaukums:

```
CALL PROC name_expr [[PARMS] (param,...)]
```

Funkcijas

Funkcijas atšķiras no procedūrām ar sākuma atslēgvārdu FUNCTION un no funkcijām tiek atgrieztas vērtības (ar atslēgvārda RETURN palīdzību). Funkcijas var izsaukt citas funkcijas, bet ne procedūras. Funkcijās var izmantos SQL vaicājumus. Ja funkcija neatgriež vērtību, tad tiek uzskatīts, ka tā atgrieza NULL vērtību.

Funkcijas definēšanas shēma:

```
FUNCTION prog_name.mod_name
[PARMS (parameter [,parameter])]
[OPTIONS (module_option [,module_option])]
[declaration_section]
executable_section
```

Funkciju izsaukums no SQL-PL notiek ar % zīmes palīdzību. Piemēram:

```
sum := %sum (a, b, c, d);
%list ('cutomers');
```

Kļūdu apstrāde

Pēc katra SQL vaicājuma tiek atgriezts rezultāts mainīgos $RC (kļūdas kods) un $RT (kļūdas paskaidrojums).

Piemērs SQL-PL kļūdu apstrādei:

```
SQL ( INSERT customer (name,firstname,city,account)
      VALUES (:cname,:cfname,:ccity,:account) );

CASE $RC OF
      200: MESSAGE := 'Customer is already registered';
      300: MESSAGE := 'Customer data erroneous';
      OTHERWISE MESSAGE := $RT;
END;
```

Paši galvenie atgrieztie kļūdas kodi ($RC)	
0	komanda veiksmīgi izpildīta
100	nav nekas atrasts
200	Atslēga jau eksistē
300	insert/update ir atteikts, jo jaunas vērtības neatbilst datu bāzes integritātei

Vēl Adabas D ļauj izķert izpildlaika (runtime) kļūdas izmantojot TRY-CATCH izteikumu. Piemērs:

```
TRY
  BEGIN
        CALL PROC do_command (...);
  END

CATCH errno OF
  16102 : ERROR := 'There is not enough memory available for this
command';
      16801 : ERROR := 'command interrupted';
END;
```

Adabas D rezervētie vārdi

ABS	DIGITS	LPAD	SHOW
ACOS	DIRECT	LTRIM	SIGN
ADDDATE	DISTINCT	MAKEDATE	SIN
ADDTIME	DOUBLE	MAKETIME	SINH
ALL	EBCDIC	MAPCHAR	SMALLINT
ALPHA	ENTRY	MAX	SOME
ALTER	ENTRYDEF	MICROSECOND	SOUNDEX
ANY	EXCEPT	MIN	SQRT
ASCII	EXISTS	MINUTE	STAMP
ASIN	EXP	MONTH	STATISTICS
ATAN	EXPAND	MONTHNAME	STDDEV
ATAN2	FIRST	NEXT	SUBDATE
AVG	FIXED	NOCACHE	SUBSTR
BINARY	FLOAT	NOCYCLE	SUBTIME
BIT	FLOOR	NOMAXVALUE	SUM
BOOLEAN	FOR	NOMINVALUE	SYSDBA
BYTE	FROM	NOORDER	TABLE
CEIL	FULL	NOROUND	TAN
CEILING	GRAPHIC	NOT	TANH
CHAR	GREATEST	NOW	TIME
CHARACTER	GROUP	NULL	TIMEDIFF
CHECK	HAVING	NUM	TIMESTAMP
CHR	HEX	NUMERIC	TIMEZONE
COLUMN	HOUR	OBJECT	TO
CONNECTED	IFNULL	OF	TOIDENTIFIER
CONSTRAINT	IGNORE	ORDER	TRANSLATE
COS	INDEX	PACKED	TRIM
COSH	INITCAP	PI	TRUNC
COT	INSERT	POWER	TRUNCATE
COUNT	INT	PREV	UCASE
CURDATE	INTEGER	PRIMARY	UNION
CURRENT	INTERSECT	RADIANS	UPDATE
CURTIME	INTO	REAL	UPPER
DATABASE	KEY	REFERENCED	USER
DATE	LAST	REJECT	USERGROUP
DATEDIFF	LCASE	REPLACE	VALUE

DAY	LEAST	RFILL	VALUES
DAYNAME	LEFT	RIGHT	VARCHAR
DAYOFMONTH	LENGTH	ROUND	VARGRAPHIC
DAYOFWEEK	LFILL	ROWID	VARIANCE
DAYOFYEAR	LINK	ROWNO	WEEKOFYEAR
DBYTE	LIST	RPAD	WHERE
DEC	LN	RTRIM	WITH
DECIMAL	LOCALSYSDBA	SECOND	YEAR
DECODE	LOG	SELECT	ZONED
DEFAULT	LOG10	SELUPD	
DEGREES	LONG	SERIAL	
DELETE	LOWER	SET	

Tabula 6 – Adabas D rezervētie vārdi

Adabas D ierobežotie vārdi

ACTION	DESCRIBE	MINUS	ROLLBACK
ADD	DISCONNECT	MODE	ROW
AND	DOMAIN	MODIFY	ROWNUM
AS	DROP	NATURAL	ROWS
ASC	EDITPROC	NO	SCHEMA
AT	END	NOWAIT	SHARE
AUDIT	ESCAPE	NUMBER	SYNONYM
BEGIN	EXCLUSIVE	OBID	SYSDATE
BETWEEN	EXECUTE	ON	TABLESPACE
BOTH	EXTRACT	ONLY	TRAILING
BUFFERPOOL	FALSE	OPEN	TRANSACTION
BY	FETCH	OPTIMIZE	TRIGGER
CASCADE	FOREIGN	OPTION	TRUE
CAST	GET	OR	UID
CATALOG	GRANT	OUTER	UNIQUE
CLOSE	IDENTIFIED	PCTFREE	UNKNOWN
CLUSTER	IN	PRECISION	USAGE
COMMENT	INDICATOR	PRIVILEGES	USING
COMMIT	INNER	PROCEDURE	VALIDPROC
CONCAT	IS	PUBLIC	VARCHAR2
CONNECT	ISOLATION	RAW	VARYING
CREATE	JOIN	READ	VIEW
CURRENT_DATE	LANGUAGE	REFERENCES	WHENEVER
CURRENT_TIME	LEADING	RELEASE	WORK
CURSOR	LEVEL	RENAME	WRITE
CYCLE	LIKE	RESOURCE	
DECLARE	LOCAL	RESTRICT	
DESC	LOCK	REVOKE	

Tabula 7 – Adabas D ierobežotie vārdi

3.2.3 Piedāvātie rīki

ADABAS D sastāv no sekojošam komponentēm:

- Rīki administrēšanai:

- Remote Control (Adcontrol) – tika izstrādāts elementārai ADABAS D administrācijai. Tas atbalsta attālinātu administrāciju vairākām SERVERDB izmantojot ērtu grafisku interfeisu.
  Šis rīks ietver sevī šādu funkcionalitāti:
  - datu bāzes inicializācija un konfigurācija;
  - datu bāzes restarts/shutdown;
  - operāciju kontrole;
  - veiktspējas kontrole;
  - dublējuma (backup) un atjaunošanas (recovery) funkcijas;
- Control (Xcontrol) – rīks, kas ir paredzēts pilnai datu bāzes administrēšanai. Pamatuzdevumi šim rīkam ir tadi paši kā Remote Control, taču šis rīks ir lokāli izmantojams.
- Ielādes rīks Load – atbalsta datu kopas un kataloga satura ielādi un izvilkšanu (extract). Ka arī tas nodrošina importa un eksporta funkciju veikšanu (datu bāzes struktūras transformācijas).
- Administrācijas rīks Domain – tas ir administratoru rīks, kas piedāvā informāciju par statiskam un dinamiskam īpašībām, kas ir definētas datu bāzes objektiem. Kā arī iespēju veidot un mainīt datu bāzes objektus (tabulas, skatījumus, sinonīmus, domeinus, indeksus, trigerus, lietotājus (user), privilēģijas, DB procedūras un DB funkcijas).

- Rīki Microsoft Windows videi:
  - ODBC draiveris – ļauj Adabas D būt pieejamam no jebkura Windows rīka, kas izmanto ODBC interfeisu. To izmanto arī UNIX rīki, lai piekļūtu datu bāzei.
  - QueryPlus (Adquery) – ļauj darboties ar Adabas D datu bāzi izmantojot SQL valodu, kā arī piekļūt datu bāzes katalogam. Šim rīkam ir Windows bāzēts interaktīvais SQL interfeiss.
  - Migrācijas rīks AccessPluss – Access datu bāzes un Adabas D datu bāzes migrācijas rīks.
- Rīki Internetam:

  - WebDB – rīks, kas nodrošina ātru savienojumu starp Web serveriem un Adabas datu bāzi. Kā arī dinamisku Web lapu ģenerēšanu.
  - DBI Perl interfeiss – nodrošina Adabas D datu bāzes pieejamību no Perl.
  - JDBC draiveris – nodrošina Adabas D datu bāzes pieejamību no JAVA, JavaScript un JAVA apletiem.
- Atvērtie interfeisi:
  - GUI Query – ļauj darboties ar Adabas D datu bāzi izmantojot SQL valodu, kā arī piekļūt datu bāzes katalogam.
  - Programmēšanas rīks SQL-PL – šis rīks ir Adabas D izstrādes vide, t.i. šeit var veidot trigerus, DB funkcijas, DB procedūras, funkcijas un procedūras.
  - Prekompilatori – ļauj iekļaut SQL pieprasījumus C/C++ un COBOL programmās.
  - Tcl/TK interfeiss – nodrošina pieeju Adabas D datu bāzei no programmēšanas valodas Tcl/TK.

4 Migrācija no ADABAS D uz Oracle

4.1 Kāpēc varētu notikt tāda migrācija

Visticamāk, ja firma nolemj pāriet no Adabas D uz Oracle, tad tā ir attīstījusies un spējīga atbalstīt Oracle datu bāzes uzturēšanu vai projektu izstrādei un uzturēšanai Adabas D datu bāzes pārvaldības sistēmas piedāvāto iespēju nepietiek, un tas ir pārāk kritiski projektam. Principa šī pāreja ir visticamākā, jo Oracle neskatoties uz lielam cenām ir spējīgāka par Adabas D.

4.1.1 Ieguvumi

Oracle patreiz ir pati pazīstamāka DBPS pasaulē, par to ir uzrakstīts liels daudzums grāmatu, Internetā var viegli atrast jebkādu informāciju par kādam problēmām. Eksistē daudz apmācoši kursi, tātad var viegli apmācīt cilvēkus kas tieši strādās ar datu bāzēm (gan administrēšanai, gan izstrādei, gan veiktspējas uzlabošanai).

Atšķirība no Adabas D Oracle jau sen uztur UNICODE, kas ir ļoti svarīgi, piemēram, Latvijas projektiem.

Oracle uzturēšanai un izstrādei ir daudz izstrādes rīkus, ar kuriem var viegli administrēt un izstrādāt Oracle datu bāzes.

4.1.2 Zaudējumi

Maksa par pašu Oracle datu bāzes vadības sistēmu ir augstāka nekā par Adabas D. Pie kam, lai normāli uzturēt Oracle datu bāzi, vajadzīgs diezgan spēcīgs aprīkojums, jo salīdzinājumā ar Adabas D Oracle visvienkāršākās datu bāzes uzturēšanai uz darbstacijas vajag 5 reiz vairāk atmiņas resursus.

Tā, ka Oracle ir vairāk attīstīta nekā Adabas D, tad tā administrēšana ir sarežģītāka. Ja kaut kas nobrūk Oracle datu bāzē, tad to ir cik vien iespējams

ātrāk pārstartēt. Tātad datu bāzes darbs 24 stundas dienā nav aktuāls Oracle datu bāzēm.

4.2 Izpildāmie soļi

4.2.1 Projekta novērtēšana

Ja tika nolemts migrēt no Adabas D uz Oracle, tad ir jāpārliecinās, ka šī migrācija nebūs pārāk dārga un laikietilpīga. Ērtākais vaids noskaidrot šis vērtības ir izmantot jau gatavu rīku kas māk izanalizēt veco datu bāzi un dot atbildi cik ilgi un cik dārgi būs migrēt. Taču diemžēl nav tāda rīka kas varētu veikt analīzi Adabas D un Oracle datu bāzes migrācijai. Tātad šo soli patreiz mums ir jāveic pašrocīgi. Tas nozīme, ņemam eksistējošo Adabas D datu bāzi un skatāmies vai

1. datu modeli, kas ir patreiz, mēs varēsim bez izmaiņām pārnest uz Oracle. Adabas D, tāpat kā Oracle, atbalsta relāciju modeļus. Tātad datu modeļa pārnešanai nevajadzētu būt pārāk sarežģītai.
2. tabulas var tikt pārnestas uz Oracle datu bāzi ar visam primāram un ārējam atslēgām, ierobežojumiem, datu tipiem, indeksiem, komentāriem. Šeit principā nekas sarežģīts nebūs. Jo gandrīz visi Adabas D datu tipi ir atbalstīti arī Oracle datu bāzē.
3. Adabas D SQL vaicājumi atšķiras no Oracle vaicājumiem un kāda apmērā (gan parastie, gan dinamiskie). Tā kā Adabas D atbalsta SQL2, bet Oracle SQL3 (kas ir papildinājums SQL2), tad šeit arī nevajadzētu būt lielam problēmām.
4. SQL-PL valodas moduli ir viennozīmīgi pārnesami uz PL/SQL. Šis punkts noteikti būs vissarežģītākais, jo Adabas D SQL-PL un Oracle PL/SQL diezgan stipri atšķiras.

Tātad sanāk, jo mazāk ir SQL-PL moduļu, jo ātrāk un lētāk notiks datu bāzes migrācija no Adabas D uz Oracle. Protams, ja projekts tika taisīts uz Adabas D formām, tad tas visas būs jāpārtaisa. Šādā gadījumā būs jāizvēlas valoda, ar kuras palīdzību varēs atbalstīt eksistējošo Adabas D formu grafisko izskatu.

Protams, var gadīties kā datu bāze projektā tika izmantota tikai datu atlasīšanai un modifikācijai, piemēram, lietojumprogrammatūra ir uztaisīta ar JAVA valodu

un tā vēršas pie datu bāzes ar standartiem SQL vaicājumiem, nevis izmanto Adabas D SQL-PL moduļus. Šādā gadījumā migrācija nebūs sarežģīta un dārga, jo pārveidot citā lietojumprogrammatūrā SQL vaicājumus ir vieglāk un ātrāk nekā pārrakstīt Adabas D procedūras un funkcijas uz Oracle procedūrām un funkcijām.

4.2.2 Shēmas un datu migrācija

Kā jau bija teikts, katram Adabas D datu tipam eksistē līdzīgs datu tips Oracle datu bāzē. Tāpat Oracle atbalsta primāras un ārējas atslēgas, pārbaudes ierobežojumus, indeksus, DEFAULT laukus, NOT NULL laukus, tabulas un kolonas komentāru veidošanu. Tātad, lai izveidot jauno datu bāzes modeli, ir jāizeksportē CREATE izteikumus, jāpiemodificē tos un tad tos var palaist uz Oracle datu bāzes.

Adabas D un Oracle tabulas īpašību neatbilstības [15]:

- Adabas D datu tips TIME netiek atbalstīts Oracle datu bāzes. Tātad ja gadīsies tabulas ar šādam kolonnām, tad būs jāpārtaisa datu tipu TIME par Oracle datu tipu VARCHAR2 (ja saglabāsim formātā HH24MMSS) vai arī var izmantot NUMBER datu tipu (taču šajā gadījumā nepaliks laika formāts, jo nāksies laiku pārvest par numuru). Protams, ja mēs izmantojam otru variantu, tad lai pārvērst laiku par numuru būs jāraksta funkcionalitāte, ka to taisīs, jo Adabas D nav tādas iebūvētas funkcijas, kas mācētu pārvest no laikā formātā uz numura formātu, tātad uz to aizies laiks un savukārt ja šādi raksti ir daudzās lielās tabulās, tad šī fāze varētu aizkavēties.
- Adabas D datu tipi VARCHAR, FIXED, FLOAT atbilst Oracle VARCHAR2 un NUMBER datu tipiem. Šis neatbilstības dēļ vajadzēs pārtaisīt tabulu veidošanas skriptus. Uz datiem tas nekā neatspoguļosies. Iespējms pazaudēt tikai pārnesot CHAR datu tipu uz Oracle, jo Adabas D šim tipam atļauj vairāk nekā Oracle.
- Ja Adabas D kādā tabulā nebija norādīta primāra atslēga, tad Adabas D automātiski saģenerēja to ar nosaukumu SYSKEY. Ja mēs gribam uztaisīt

normalizētu datu bāzes relācijas modeļi, tad tabulu veidošanas skriptos būs jāpievieno jaunu lauku (parasti ar NUMBER datu tipu) un primāras atslēgas pievienošanu. Uz datiem tam nevajadzētu atspoguļoties, jo, ja tabulā nebija definēta primāra atslēga, tad to citās tabulās arī neizmanto.

- Adabas D ir speciāls datu tips SERIAL vai AUTOINCREMENT, kas ir FIXED datu tipa paplašinājums un ļauj saglabāt datus augošā secībā, t.i. ja INSERT teikuma mēs norādīsim kādu numuru, tad tam ir jābūt lielākam par iepriekšējo lielāku numuru. Oracle tāda datu tipa nav, taču ir atbalstīta SEQUENCE lietošana, kas principā ir tas pats. Tātad ja tabulā ir kolona ar tipu SERIAL, tad to samainām uz NUMBER datu tipu un kad dati ir pārnesti tabulā, tad arī izveidojam uz tas kolonas SEQUENCE, lai turpmāki dati tiktu ielikti tabulā ar pieaugošu secību un lai nesabojāt tos datus, kas bija patreiz.
- Adabas D atbalsta ierobežotus identifikatorus (delimited identifiers), t.i., piemēram, nosaucot kolonu `"CNO "` pēc tam var vērsties pie tas vienkārši rakstot CNO. Adabas D ierobežotie identifikatori nav reģistrjutīgie. Oracle arī var izmantot ierobežotus identifikatorus, taču tie ir reģistrjutīgie, t.i. ja mēs nosauksim kolonu `"CNO "`, tad pēc tam mēs nevarēsim SQL teikumos izmantot to kā `"CNO"` vai `"cno"`. Tāpēc labāk, ja tas nav rezervēts Oracle vārds (rezervētie Oracle vārdi ir pārskaitīti Tabula 5), pārvērst Adabas D ierobežotus identifikatorus par regulāriem identifikatoriem.
- Adabas D pēc datu tipa norādīšanas var atstāt tukšumzīmes, taču Oracle tas nav atļauts. Tāpēc pēc datu tipa norādīšanas atstarpes ir jāizņem.
- Adabas D un Oracle aizvietojamie datu tipi

Adabas D	Oracle
`CHAR (N)`, kur N <= 4000	`CHAR(N)`, kur N <= 2000 (sinonīms `CHARACTER`)
`VARCHAR(N)`, kur N <= 4000	`VARCHAR(N)`, kur N <= 4000 (sinonīmi `STRING` un `VARCHAR`)
`BOOLEAN`, kur vērtības TRUE vai FALSE	Tabulās tādu datu tipu var aizvietot ar

	CHAR(1) ar definētām vērtībām. PL/SQL tādu datu tipu var izmantot, vērtības arī TRUE vai FALSE.
DATE, kur noklusēts formāts ir YYYYMMDD	DATE, kur noklusēts formāts ir YYYY-MM-DD Ja datuma formāti nesakrīt, tad ir jāizmanto Oracle funkcija TO_DATE.
TIME, kur noklusēts formāts ir HHHHMMSS	-
TIMESTAMP, kur noklusēts formāts ir YYYYMMDDHHMMSSmmmμμμ	TIMESTAMP, kur noklusēts formāts ir YYYY-MM-DD HH:MM:SS.mmm Mikrosekundes šajā datu tipā nerādās.
FIXED(N,M), kur maksimāli var būt 18 cipari	NUMBER(N,M), kur maksimāli var būt 38 cipari. (sinonīmi DEC, DECIMAL, NUMERIC, INTEGER, INT, SMALLINT, DOUBLE, PRECISION, FLOAT, REAL
SERIAL/AUTOINCREMENT	-
FLOAT(N), kur maksimāli var būt 18 cipari	FLOAT(N), kur maksimāli var būt 38 cipari.
LONG, kur maksimāli var būt 2.1 GB	LONG, kur maksimāli var būt 2 GB, CLOB, NCLOB, BLOB, BFILE, XMLType – 4 GB.
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT	FLOAT
DOUBLE	DOUBLE
PRECISION	PRECISION
REAL	REAL
LONG VARCHAR	CLOB
BYTE	RAW

Tabula 8 - Adabas D un Oracle aizvietojamie datu tipi

Adabas D ir speciāls rīks LOAD, kas var eksportēt datu bāzes struktūru un datus atsevišķos failos ar veidošanas skriptiem un tabulu , kurus vēlāk var palaist. Taču

gan veidošanas skripti (lietotājiem, tabulām, indeksiem, u.t.t.), gan datu ielādēšanas skripti ir izveidoti speciāli to palaišanai uz Adabas D. Tas nozīme, ka būs jāpārveido visus failiņus ar šiem skriptiem Oracle datu bāzes formātā [16].

Piemērs tabulas veidošanas skriptam, ko uzģenerēja LOAD rīks, izmantojot CATALOGEXTRACT TABLE izteikumu:

```
CREATE TABLE "SQLTRAVEL00"."CUSTOMER"           (
"CNO                " FIXED      (4,0) ,
"TITLE              " CHAR       (7) ASCII    ,
"FIRSTNAME          " CHAR       (10) ASCII    ,
"NAME               " CHAR       (10) ASCII    NOT NULL,
"ZIP                " CHAR       (5) ASCII    ,
"ADDRESS            " CHAR       (25) ASCII    NOT NULL,
"PHONENO            " CHAR       (15) ASCII
, CONSTRAINT "CNO_DOM           " CHECK
 CNO BETWEEN 1 AND 9999
, CONSTRAINT "TITLE             " CHECK
TITLE IN ('Mr', 'Mrs', 'Company')
, PRIMARY KEY ("CNO"
))
/ *
```

Lai būtu iespējams palaist šo skriptu Oracle, to ir jāpārveido šādi:

```
CREATE TABLE SQLTRAVEL00.CUSTOMER(
CNO             NUMBER(4,0),
TITLE           VARCHAR2(7),
FIRSTNAME       VARCHAR2(10),
NAME            VARCHAR2(10)    NOT NULL,
ZIP             VARCHAR2(5),
ADDRESS         VARCHAR2(25)    NOT NULL,
PHONENO         VARCHAR2(15)
, CONSTRAINT CNO_DOM CHECK
 (CNO BETWEEN 1 AND 9999 )
, CONSTRAINT TITLE CHECK
(TITLE IN ('Mr', 'Mrs', 'Company') )
, PRIMARY KEY (CNO)
)
TABLESPACE  TABLSP1
/
```

Izņemot skriptu pārveidošanu vajadzētu izanalizēt tabulu references un uztaisīt atsevišķus failiņus, kuros būs atsevišķi loģiski sakārtoti tabulu, sinonīmu, GRANT, ārējo atslēgu, indeksu un skatījumu veidošanas skripti, kā arī datu ielādēšanas skripti, jo Adabas D ģenerē šos visus skriptus vienā failā un nesakārtotus, un palaižot tos kopā, var rasties daudz kļūdu.

Kad visi skripti ir gatavi, tos var viegli palaist Oracle datu bāzē izmantojot rīkus SQL*PLUS un SQL*LOADER.

Instalēšanas secība:

1. Tabulas
2. Sinonīmi
3. Datu ielikšana
4. Indeksi
5. Sekvences
6. GRANT
7. Skatījumi
8. Datu bāzes saites

4.2.3 Biznesa loģikas migrācija

Šajā fāzē vajadzēs

- izeksportēt visus SQL-PL moduļus;
- izanalizēt SQL-PL un PL/SQL valodas atšķirības;
- pārtaisīt visus SQL-PL moduļus uz PL/SQL moduļiem;
- uzinstalēt gatavus PL/SQL moduļus Oracle datu bāzē.

Ar Adabas D rīku LOAD nav iespējams izeksportēt datu bāzes SQL-PL moduļus. To var mēģināt darīt SQL-PL rīkā, taču eksportēt nāksies katram lietotājam atsevišķi.

Kad visas DB procedūras, DB funkcijas, procedūras, funkcijas un trigeri ir izeksportēti, sakas to analīze. Ir jāsalīdzina SQL-PL un PL/SQL valodu struktūras un to atšķirības, kā arī jāzina kā var pārveidot konkrētas struktūras.

Zemāk ir aprakstīts kā jāizskatās Oracle PL/SQL programmām (Tabula 9).

Adabas D	Oracle
Bāzes elementi	
Komentāri	

`/* komentārs`	`-- komentārs`
	vai
	`/* liels`
	`Komentārs */`

Nosaukumi	
Moduļu, mainīgu un datu bāzes objektu nosaukumus no Adabas D SQL-PL var pārnest uz Oracle PL/SQL bez izmaiņām. Vienīgais ir jāuzmanās no Oracle rezervētiem vārdiem (sk. Tabula 5). Ja Adabas D kāds modulis, objekts vai mainīgais tika nosaukts kā Oracle rezervētais vārds, tad ir jāizskata parēji saistītie objekti (vai mainīgie, vai moduļi) un uz Oracle to jāpārnes ar nākamo brīvo nosaukumu. Piemērs:	
`PROC customer.compress PARMS (a)` `...`	`PROCEDURE compress1 (a NUMBER) AS` `...`
Literāli	
Numuru un simbolu literāli sakrīt Adabas D un Oracle. Taču Adabas D vēl atbalsta heksadecimālus literāļus, bet Oracle neatbalsta. Tātad ja SQL-PL modulī satiekam kādu heksadecimālo skaitli, tad to nāksies pārvērst Oracle atbilstoša formātā (numuros vai simbolos).	
Mainīgie	
Adabas D ir 3 mainīgo veidi, t.i. globālie, lokālie-dināmiskie un lokālie-statiskie. Kā arī mainīgie var būt skalārie (satur 1 vērtību) un vektori (satur vairākas vērtības). Tātad ja mēs programmā pēc moduļa galvas redzam atslēgvārdu VAR un tiek definēti skalārie mainīgie, tad ir no moduļa konteksta jānoskaidro kāds ir šim mainīgajam tips. To var noskaidrot no piešķiršanas operatoriem vai no SQL vaicājuma.	

`VAR a, b, c;`
`...`
`@a := 'asdad';`
`@b := 12;`
`@c := DATEDIFF(max_departure,`
`min_arrival);`
	`DECLARE`
	`  a VARCHAR2(255);`
	`  b NUMBER;`
	`  c DATE;`
	`BEGIN`
	`  a := 'asdad'; b := 12; c :=`
	`max_departure - min_arrival;`

`VAR max_free;`
`...`
`SQL (SELECT max_free`
`  INTO :@max_free`
`  FROM sqltravel10.room`
`  WHERE cno = 1);`
	`DECLARE`
	`  max_free`
	`sqltravel10.room.max_free%TYPE;`
	`BEGIN`
	`  SELECT max_free`
	`  INTO max_free`

	FROM sqltravel10.room WHERE cno = 1;
Tā kā Oracle neatbalsta masīvus kā tādus, tad vektora mainīgus ir jāpārveido par Oracle kolekcijām. Ja ir zināms vektora maksimālais elementu skaits, tad pārveidosim par Oracle kolekciju VARRAY, savādāk par līdztabulu (nested table).	
VAR emp_tab();	DECLARE TYPE EmpTabTyp IS TABLE OF INTEGER INDEX BY PLS_INTEGER; emp_tab EmpTabTyp;
Ja mainīgais netiek deklarēts pašā modulī, tad tas ir globālais mainīgais. Šādā gadījumā vajag to definēt pakotnes galvā. Deklarēšanas principi ir tādi paši kā ar lokāliem mainīgiem. Tas ir, ir jānoskaidro mainīga tips un definēt to attiecīgi tālākai izmantošanai.	

IF cnt > 0 THEN single := cnt
ELSE single := NULL;
	--pakotnes galvā
	cnt NUMBER;
	single NUMBER;
	
	--pakotnes ķermenī
	IF cnt > 0 THEN
	single := cnt;
	ELSE
	single := NULL;
	END IF;

SQL (SELECT cst_inv_res(firstname, name) FROM SQLTRAVEL00.CUSTOMER ORDER BY name);	--pakotnes galvā TYPE cst_inv_res IS TABLE OF SQLTRAVEL00.CUSTOMER%ROWTYPE INDEX BY PLS_INTEGER; firstname cst_inv_res.firstname; name cst_inv_res.name; --pakotnes ķermenī SELECT firstname, name INTO firstname, name FROM SQLTRAVEL00.CUSTOMER ORDER BY name;
Vēl Adabas D atļauj padotos parametrus (ar atslēgvārdu INOUT vai bez atslēgvārda) mainīt, tātad tie arī ir moduļa mainīgie. Oracle moduļos var mainīt parametrus, kar ir ar atslēgvārdu OUT vai IN OUT. Parametri arī var būt skalārie vai vektori, globālie, lokālie-dināmiskie vai lokālie-statiskie. To pārnešana uz Oracle notiek līdzīgi parasto mainīgu pārnešanai.	
Izteiksmes	
Aritmētiskie, virknes un loģiskie izteikumi Adabas D un Oracle lielākoties sakrīt. Atšķirības:	

- aritmētiskās izteiksmēs var izmantot DB funkcijas (`a := %vsum(b,c,d);`), šajā gadījumā vajag vienkārši izņemt % zīmi;
- virknes izteiksmēs konkatenācijas zīmi & ir jāaizvieto ar || zīmi;
- loģiskās izteiksmēs, kad tiek izmantota salīdzināšana (`LIKE 'S*'`), * zīmes ir jāaizvieto ar % zīmēm.

Kontroles struktūras

IF izteikums

```
IF <boolean expr>
THEN <compound>
[ELSE <compound>];
```

```
IF <boolean expr>
THEN <compound>
[ELSE <compound>]
END IF;
```

CASE izteikums

```
CASE <expr> OF
 <value spec list> : <compound> ;
 [<value spec list> : <compound> ;]
 [OTHERWISE <compound>]
END;
```

```
CASE <expr>
 WHEN <value spec list> THEN
<compound> ;
 [WHEN <value spec list> THEN
<compound> ;]
 [ELSE <compound>]
END CASE;
```

vai

```
CASE
 WHEN <search_condition> THEN
<compound> ;
 [WHEN <search_condition> THEN
<compound> ;]
 [ELSE <compound>]
END CASE;
```

REPEAT izteikums

```
REPEAT <stmt>;
  ...
UNTIL <boolean expr>;
```

```
WHILE <boolean expr> LOOP
  <stmt>;
  ...
END LOOP;
```

WHILE izteikums

```
WHILE <boolean expr> DO <compound>
```

```
WHILE <boolean expr> LOOP
  <stmt>;
  ...
END LOOP;
```

FOR izteikums

```
FOR <variable> := <expr> TO <expr>
DO <compound>
```

vai

```
FOR <variable> := <expr> DOWNTO
<expr> DO <compound>
```

```
FOR <variable> IN [REVERSE]
 <expr>..<expr> LOOP
  <compound>
END LOOP;
```

vai

```
FOR <variable> IN [REVERSE]
 (<select_stmt>) LOOP
  <compound>
END LOOP;
```

SKIP izteikums	

`SKIP label;`
`  ...`
`label : statement;`
	`GOTO label;`
	`  ...`
	`<<label>>`
	`statement;`

RETURN izteikums	
Šis izteikums sakrīt gan Adabas D, gan Oracle.	
STOP izteikums	
STOP izteikums paredzēts moduļa darba apstādināšanai un visu SQL darbību atritei (rollback), atgriežot iziešanas vērtības. To var mēģināt aizstāt ar ROLLBACK un RETURN izteikumu, arī atgriežot vērtību un pēc konkrēta moduļa izsaukuma pārbaudīt tieši uz atgriezto vērtību.	

STOP [(<expr> [, <expr>])]	Funkcijās:
	`ROLLBACK;`
	`RETURN (-1); -- pārbaude uz -1`
	Procedūrās:
	`PROCEDURE a (OUT v_errmsg NUMBER)`
	`AS`
	`...`
	`ROLLBACK;`
	`v_errmsg := -1; -- pārbaude uz -1`
	`RETURN;`

Moduļu izsaukšana	
Procedūru izsaukumi	

`CALL PROC display PARMS (cno, resdat);`	`display (cno, resdat);`
	vai
	`display (cno => cno, resdat => resdat);`

`SWITCH reservation CALL PROC list;`	Oracle nav paredzēts izmantot SWITCH (jo vienmēr pēc katras procedūras pabeigšanas vai kļūdas gadījumā mēs atgriežamies uz vietu, kur tika izsaukta procedūra), tāpēc izmatosim tādu pašu procedūru izsaukšanu kā iepriekš.
Funkciju izsaukumi	

`sum := %sum (a, b, c, d, e);`	`v_sum := vsum (a, b, c, d, e);`
	vai
	`v_sum NUMBER := vsum(a, b, c, d, e);`

`%liste ('customerlist');`	Šādi funkciju izsaukumi nav atļauti Oracle, tāpēc to vienalga ir jāatgriež kādam mainīgam.
DB procedūru izsaukumi	

`CALL DBPROC [[<owner>.]<progname>.]<dbproc name> [PARMS (<param>,...] [WITH COMMIT]`
vai
`SQL ( DBPROC[EDURE] [[<owner>.]<progname>.]<dbproc name> [(<host var>,...)] [WITH COMMIT] )`
	Tā kā visas DB procedūras tiks aizvietoti ar Oracle procedūrām, tad izsaukums būs tāds pats kā procedūru izsaukums.

DB funkciju izsaukumi	

`SQL (SELECT fupper(name) FROM SQLTRAVEL00.CUSTOMER WHERE cno = 1);`	`SELECT fupper(name) INTO v_name FROM SQLTRAVEL00.CUSTOMER WHERE cno = 1;`
	vai
	`SELECT name INTO v_name FROM SQLTRAVEL00.CUSTOMER WHERE cno = 1; v_name := fupper(v_name);`

Trigeru izsaukumi	
Līdzīgi kā Adabas D trigeris tiek izsaukts, kad tiek veikta darbība uz tabulu, kas ir definēta trigerī.	
SQL izmantošana	
Piekļuve datu bāzei	
Adabas D visi SQL vaicājumi apzīmējas ar atslēgvārdu SQL un ir ierakstīti apaļās iekavās (t.i. nedinamiskie SQL vaicājumi), bet Oracle SQL vaicājumi ir bez palīg atslēgvārdiem. Adabas D SELECT vaicājumam nav obligāti uzreiz atgriezt vērtību kādā mainīga, to var izdarīt vēlāk ar FETCH palīdzību, taču Oracle PL/SQL tas ir obligāti, t.i. SELECT teikums nevar būt bez INTO.	
Datu manipulācijas:	
Gan Adabas D, gan Oracle izmanto INSERT, UPDATE, DELETE un SELECT komandas.	

`SQL (INSERT [INTO] <table name> [(<column name>,...)] VALUES (<extended expression>,...))`
vai
`SQL (INSERT [INTO] <table name>`
	`INSERT INTO <table name> [(<column name>,...)] VALUES (<extended expression>,...)`
	vai
	`INSERT INTO <table name> [(<column`

[(<column name>,...)] <query expression>) vai SQL (**INSERT** [INTO] <table name> SET <set insert clause>,...)	name>,...)] <query expression>
SQL (**UPDATE** [OF] <table name> [<reference name>] (<column>,...) VALUES (<extended value spec>,...) [KEY <key spec>,...] [WHERE <search condition>]) vai SQL (**UPDATE** [OF] <table name> [<reference name>] SET <column name> = <extended expression> [| <subquery>],... [KEY <key spec>,...] [WHERE <search condition>]) vai SQL (**UPDATE** [OF] <table name> [<reference name>] (<column>,...) VALUES (<extended value spec>,...) WHERE CURRENT OF <result table name>) vai SQL (**UPDATE** [OF] <table name> [<reference name>] SET <column name> = <extended expression> [| <subquery>],... WHERE CURRENT OF <result table name>)	UPDATE <table name> SET <column name> = <extended expression> [| <subquery>],... [WHERE <search condition>]
SQL (**DELETE** [FROM] <table name> [<reference name>] [KEY <key spec>,...] [WHERE <search condition>])	DELETE [FROM] <table name> [WHERE <search condition>]
SQL (**SELECT** [DISTINCT] <select column>, ... [INTO :<variable>, ...] FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>])	SELECT [DISTINCT] <select column>, ... INTO <variable>, … FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]
Ja SQL teikumā notiek salīdzināšana ar SQL-PL mainīgo (cno = :@cno), tad „:” ir jāizņem, jo Oracle pats saprot kā notiek salīdzināšana ar mainīgo.	
Transakciju kontrole:	
COMMIT [WORK] [KEEP <lock statement>];	COMMIT;
ROLLBACK [WORK] [KEEP <lock	ROLLBACK;

statement>]	
SUBTRANS BEGIN| SUBTRANS END| SUBTRANS ROLLBACK	Adabas D šis transakcijas tiek izmantotas DB procedūrās, tā kā Oracle būs parastas procedūras un funkcijas un atgriešanas uz moduli, kas izsauca konkrētu procedūru (funkciju) būt vienmēr, tad apakštransakcijas Oracle nav vajadzīgi.
SQL Operatori	
ALL, BETWEEN, EXISTS, IN, IS NULL, LIKE, SOUNDS, ESCAPE, SOME, ANY	ALL, BETWEEN, EXISTS, IN, IS NULL, LIKE, SOME, ANY
Kursori	
DECLARE <result table name> CURSOR FOR SELECT [DISTINCT] <select column>, ... FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]	CURSOR <result table name> IS SELECT [DISTINCT] <select column>, ... FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>];
Kolekcijas	
SELECT [DISTINCT] <result table name> (<select column>,...) FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]	SELECT [DISTINCT] BULK COLLECT INTO <select column>,... FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]
Dinamisko SQL izmantošana	
Dinamiskais SQL priekšraksts tiek veidots no SQL vaicājumiem, transakcijām, izmantojot atdalošo izteiksmi un mainīgus. Tā var ērti piekļūt datu bāzei.	
SQL [<dbname>] EXECUTE IMMEDIATE <expr> SQL EXECUTE IMMEDIATE <expr> USING <expr list> SQL EXECUTE <name> USING <expr list> SQL PREPARE <name> FROM <expr> SQL DESCRIBE <name> INTO <vector slice>	EXECUTE IMMEDIATE <expr> Taču dinamiskus SQL vaicājumus pirms pārnešanas ir jāizanalizē, t.i. jābūt pārliecinātam, ka tādu SQL vaicājumu ir iespējams palaist Oracle. Tātad ja vajag, tad <expr> arī ir jāpārveido uz attiecīgu Oracle formātu.
Kļūdu apstrāde	

Pēc katra SQL vaicājuma vai SQL-PL moduļa izpildes tiek atgriezta vērtība $RC un $RT parametros. Un ja šī vērtība nav vienāda ar 0, tad tas nozīme, ka notika kļūda. Pēc $RC (kļūdas kods) var noskaidrot kas tā bija par kļūdu. $RT parametrs satur kļūdas paskaidrojumu angļu valodā.	Oracle par kļūdām tiek uzskatīti izņēmumi. Lai dabūt un apstrādāt izņēmumu visi SQL vaicājumi tiek ielikti tā saucama anonīma blokā un pēc izteikuma aplamas izpildes tiek apstrādāti izņēmumi. Ir Oracle definētie izņēmumi un tos iespējams definēt arī pašiem.

`SQL (DELETE customer`
`  KEY cno = :cno);`
`IF $RC = 0`
`  THEN MESSAGE := 'This entry has been deleted!'`
`  ELSE MESSAGE := 'rejected ('& $RC &')';`
	`BEGIN`
	`  DELETE * FROM customer WHERE cno = cno;`
	
	`  dbms_output.put_line('This entry has been deleted!');`
	`EXCEPTION`
	`  WHEN OTHERS THEN`
	`  dbms_output.put_line('rejected ('|| SQLERRM ||')');`
	`END;`

`TRY`
`  BEGIN`
`  CALL PROC do_command (...);`
`  END`
`CATCH errno OF`
`  16102 : ERROR := 'There is not enough memory available for this command';`
`  16801 : ERROR := 'Command interrupted';`
`END;`
	`BEGIN`
	`  do_command (... ,errno); --errno IN OUT EXCEPTION`
	`EXCEPTION`
	`  WHEN errno THEN`
	`  dbms_output.put_line('Error: '||errno);`
	`  WHEN OTHERS THEN`
	`  dbms_output.put_line('Error: '||SQLERRM);`
	`END;`

SQL-PL moduļu pārveidošana	
DB procedūras un procedūras tiks pārveidoti par Oracle procedūrām. DB funkcijas un funkcijas tiks pārveidoti par Oracle funkcijām. Trigeri paliek par trigeriem, taču Oracle PL/SQL formātā. Visas iebūvētas Adabas D funkcijas, kas nav starp Oracle iebūvētam funkcijām būs jātaisa pašrocīgi. Piemērs iebūvētai funkcijai:	

`DATEDIFF ('19990101','19980101');`
`/* returns 365`
	`CREATE OR REPLACE FUNCTION datediff (date1 DATE, date2 DATE)`
	`RETURN NUMBER IS`
	`BEGIN`
	`  RETURN (date1 - date2);`
	`END;`

No visiem moduļiem, kas atrodas vienā Adabas D programmā, ir jāuztaisa Oracle pakotne ar tādu pašu nosaukumu, kāds ir Adabas D programmai. Tā būs	

ērtāk dot lietotājiem tiesības uz konkrētām procedūrām/funkcijām izpildīšanu. Tā kā mēs gribēsim visus pārveidotus moduļus uzinstalēt jaunā datu bāze, tad katras atsevišķas procedūras vai funkcijas, pakotnes vai trigera sakumā būs jāieraksta **CREATE OR REPLACE**.

Trigerus Adabas D laiž tikai pēc INSERT, UPDATE vai DELETE darbības, tāpēc visus Adabas D trigerus Oracle ir jāuztaisa kā AFTER trigerus.

Piemēri trigerim un funkciju pārveidošanai:

```
TRIGGER triggs20.cust_full_name( IN NEW.cust_group CHAR(1),
                                 IN NEW.name VARCHAR(60),
                                 IN NEW.first_name VARCHAR(30),
                                 IN NEW.last_name VARCHAR(30))

IF :new.cust_group = 'J' THEN
  IF :new.NAME IS NOT NULL THEN
    :new.full_name := :new.NAME;
  END IF;

ELSIF :new.cust_group = 'F' THEN
  IF :new.first_name IS NOT NULL THEN
    :new.full_name := :new.first_name;
    IF :new.last_name IS NOT NULL THEN
      :new.full_name := :new.full_name || ' ' || :new.last_name;
    END IF;
  ELSIF :new.last_name IS NOT NULL THEN
    :new.full_name := :new.last_name;
  END IF;

END IF;
ENDMODULE
```

```
CREATE OR REPLACE TRIGGER cust_full_name
AFTER INSERT OR UPDATE OF cust_group, name, first_name, last_name
ON customer
REFERENCING NEW AS NEW OLD AS OLD
BEGIN
  IF :new.cust_group = 'J' THEN
    IF :new.NAME IS NOT NULL THEN
      :new.full_name := :new.NAME;
    END IF;

  ELSIF :new.cust_group = 'F' THEN
    IF :new.first_name IS NOT NULL THEN
      :new.full_name := :new.first_name;
      IF :new.last_name IS NOT NULL THEN
        :new.full_name := :new.full_name || ' ' || :new.last_name;
      END IF;
    ELSIF :new.last_name IS NOT NULL THEN
```

```
      :new.full_name := :new.last_name;
    END IF;

  END IF;
END;
/
```

```
FUNCTION mailbox_eng.new_mno;

SQL (SELECT (max(mno) + 1) INTO :@mno FROM mail.messages);
IF @mno IS NULL
THEN
     @mno := 1;
IF @mno = 1000000
THEN BEGIN
     @mno := 0;
     REPEAT
        @mno := @mno + 1;
        SQL (SELECT DIRECT mno INTO :@dummy
             FROM mail.messages KEY mno = :@mno);
     UNTIL $RC = 100;
     END;
RETURN (@mno);
ENDMODULE
```

```
CREATE OR REPLACE FUNCTION new_mno RETURN NUMBER AS
DECLARE
  mno mail.messages%TYPE;
BEGIN
  BEGIN
    SELECT max(mno) + 1
    INTO mno
    FROM mail.messages;

    IF mno = 1000000 THEN
      mno := 1;
      FOR i INTO (SELECT mno
                  FROM mail.messages
                  WHERE mno = mno) LOOP
        mno := i.mno + 1;
      END LOOP;
    END IF;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      mno := 1;
    WHEN OTHERS THEN
      NULL;
  END;

  RETURN (mno);
END;
```

Tabula 9 – Adabas D SQL-PL un Oracle PL/SQL valodu salīdzinājums, iespējami risinājumi

Instalēšanas secība:

1. Pakotņu galvas (header)
2. Trigeri
3. Procedūras
4. Funkcijas
5. Pakotņu ķermeni

Pakotņu ķermeņus, procedūras un funkcijas ir jāinstalē tā, lai instalējamā modulī nebūtu vēl neuzinstalēto moduļu izsaukumus. Instalēt izveidotos PL/SQL moduļus var ar SQL Plus rīku.

4.2.4 Iegūtas datu bāzes testēšana

Tikko datu bāze ir pārnesta un viss ir uzinstalēts, tad ir jātestē vai tiešam viss strādā, tā kā tas bija agrāk. Vislabākais variants būtu izstrādāt programmas, kas pieslēdzas abām datu bāzēm un vienlaicīgi vai pēc kārta izpilda vienas un tas pašas darbības, t.i. palaiž procedūras un funkcijas, ievieto, atjauno vai dzēš vienādus datus tabulās. Un tikko darbība ir izpildīta, tad salīdzina rezultātus. Galvenais lai rezultāts, kuru mēs dabūjam vecā datu bāze, ir tāds pats kā jaunā.

4.2.5 Lietojumprogrammatūras migrācija

Lietojumprogrammatūras ar datu bāzes atbalstu var būt divos veidos, t.i. uzprogrammētas ar citu programmēšanas valodu vai uz datu bāzes valodas pamatā. Pirmais variants ir tad, kad datu bāzes ir vajadzīga tīri datu manipulācijām un parēja projekta daļa ir uzprogrammēta atsevišķi. Kā piemērs ir interneta veikals, labs dizains un visa informācija par precēm glabājas datu bāzē. Šajā variantā migrāciju veikt ir vieglāk, jo principā var viegli noskaidrot kādas vietās notiek sadarbība ar datu bāzi un ja vajag pielabot tas. Tas ir, mūsu uzdevums ir atrast vai nu uzreiz SQL vaicājumus datu bāzei, vai SQL-PL programmas izsaukumus un izlabot uz Oracle atbilstošiem vaicājumiem un jau izveidoto PL/SQL moduļu izsaukumiem. Grūtības var būt, ja, piemēram, pēc SQL-PL programmas izpildes tiek atgriezti $RC un $RT vērtības un pēc tām tie tiek apstrādāti un no tā seko citas darbības. Tā kā Oracle PL/SQL programmas neatgriež nekādas specifiskas vērtības, izņemot kad programmas izpildes laikā notika vai tika speciāli izraisīta kāda kļūda, tad visas $RC vērtību apstrādes

nāksies pārveidot atbilstoša Oracle apstrāde, t.i. ja programma izpildījās bez kļūdām, tad darīt to pašu kas bija, ja $RC = 0.

Taču ir iespēja kā projekts bija veidots uz Adabas D formām un atskaitēm (reports), šajā gadījumā būs jāpārveido visu formas un atskaišu funkcionalitāti uz citu programmēšanas valodu. Un šāda migrācija jau pavisam nav tik primitīva. Jo atkarība no izvelētas valodas vajadzēs atkal taisīt SQL-PL un citas valodas sintaksisko un semantisko analīzi. Bet uzdevums manam bakalaura darbam nav pilnīgi migrēt Adabas D lietojumprogrammatūru ar datu bāzi, tāpēc šajā darba netiks izskatīta detalizēta Adabas D lietojumprogrammatūras migrācija.

4.2.6 Testēšana, integrēšana un izvietošana

Sakuma visu ir jāpalaiž, t.i. jāpārliecinās vai mums ir pareizs savienojums ar jaunu datu bāzi, vai kādi no izmainītiem (jaunizveidotiem) moduļiem nepalika nederīgi (invalid). Pēc tam tāpat kā iepriekš pārbaudām vai visi lietojumprogrammatūru moduļi strādā tāpat kā iepriekš, vai iepriekš sadefinētas darbības rezultāts ir tāds pats, vai netika pazaudētas kādas lietojumprogrammatūras iespējas.

4.2.7 Veiktspējas optimizēšana

Šo fāzi, ja tas nav kritiski, var veikt jau pēc projekta palaišanas uz jaunās datu bāzes. Bet principā, Oracle ir ļoti daudz iespēju kā uzlabot veiktspēju. Piemēram, ja ir redzams, ka tabulā ir liels datu apjoms un ar kādu laika intervāli datu daudzums tabulā palielinās, tad šo tabulu var sadalīt uz vairākiem nodalījumiem (partitions). Piemēram, ja tabulā mums glabājas klientu rēķini un ir zināms kā katru mēnesi to skaits palielinās uz „aktīvo" klientu skaitu plus vēl citi neregulārie rēķini, tad šo tabulu ir vērts sadalīt pēc datumiem (piemēram, pa kvartāliem), tad sameklēt pēdējus aktīvus rēķinus klientam pa vienu tabulas nodalījumu būs ātrāk nekā pa visu tabulu. Vēl Oracle atšķirībā no Adabas D ir ļoti attīstīti SQL vaicājumu norādījumi (hints), izmantojot tos var diezgan palielināt vaicājuma ātrdarbību. Bet tas jau ir sīkums, salīdzinājumā ar pārējām fāzēm.

4.3 Secinājumi

Datu bāzes migrācija no Adabas D uz Oracle ir laba ar:

- Ātrdarbības uzlabošanas iespēju ir vairāk nekā Adabas D;
- Literatūras pieejamība;
- UNICODE atbalsts;
- Plašā izstrādes rīku izvēle;
- Papildus datu bāzes iespējas, kas nebija Adabas D (objektu tipi, kolekcijas, transakciju kontrole u.t.t.);

Datu bāzes migrāciju no Adabas D uz Oracle ir slikta ar:

- Oracle uzturēšana ir daudz reiz dārgāka par Adabas D uzturēšanu;
- Administrēt un uzturēt pašu datu bāzi ir grūtāk nekā Adabas D, dēļ Oracle iespējam, kas nav Adabas D;
- Bāzes nobrukšana var pievest datu bāzes apstādināšanas vai pārlogošanas;
- Oracle tomēr dažu Adabas D datu tipus neatbalsta, piemēram, TIME datu tips, tādēļ var sabojāties dati vai funkcionalitāte, kas ir saistīta ar tiem;

Tātad ja ņemt vērā kā jebkura komerciāla DBPS maksa naudu, cita vairāk, cita mazāk un ar plašākam Oracle iespējam ir daudz pamatojuma domāt kā projekta tālāka izvēršanas notiks diezgan veiksmīgs, tāpēc ir pamatojums apgalvot, ka migrācija no Adabas D uz Oracle ir labs daudzu problēmu risinājums.

5 Migrācija no Oracle uz ADABAS D

5.1 Kāpēc varētu notikt tāda migrācija

Visticamāk, ja firma nolemj pāriet no Oracle uz Adabas D, tad laikam visspēcīgākais iemesls ir paša Oracle datu bāzes vadības sistēmas un uzturēšanas izmaksas ir pārāk lielas un tas nav šai firmai pa spēkam. Un apsverot ieguvumus un zaudējumus tiek nolemts tomēr pāriet uz citu DBPS (šajā gadījumā uz Adabas D). Tālāk tiek pārskaitīti galvenie ieguvumi un zaudējumi, ja notiks šāda migrācija.

5.1.1 Ieguvumi

Maksa par Adabas D datu bāzes vadības sistēmu ir krietni zemāka nekā par Oracle.

Adabas D datu bāze „noēd" mazāk aparatūras resursus salīdzinot ar Oracle datu bāzi, tāpēc nav vajadzīgs tik spēcīgs aprīkojums, lai datu bāze varētu normāli strādāt.

Adabas D ir vienkāršāka struktūra un vieglāk saprotama. Piemēram, uztaisīt un konfigurēt jaunu datu bāzi uz Adabas D ir krietni vieglāk nekā ar Oracle.

Adabas D piedāvā 24 stundas darbu un ja kaut kas nobrūk, tad nav nepieciešams apstādināt datu bāzi.

5.1.2 Zaudējumi

Nav tik daudz literatūras par pašu Adabas D, jo tas nav tik izplatīta datu bāzes vadības sistēma. Un tas nozīme, ja rodas kādas neskaidrības vai problēmas, tad uzzināt kā jārisina konkrēta problēma nav tik vienkārši kā ar Oracle.

Ar pašreizējo Adabas D versiju, ja iepriekšēja datu bāzes izmantoja UNICODE, būs problēmas (piemēram, pārnesot datus ar latviešu burtiem), jo Adabas D

13.01 atbalsta tikai ANSII kodējumus. Taču pēc šī gada trešā kvartālā iznāks Adabas D jauna versija ar UNICODE atbalstu.

Adabas D vēl nav papildus izstrādes rīkus (piemēram, kā Oracle TOAD, SQL Navigator u.c.), tikai tie, kurus piedāvā pats Adabas D.

5.2 Izpildāmie soļi

5.2.1 Projekta novērtēšana

Ja tika nolemts migrēt no Oracle uz Adabas D, tad ir jāpārliecinās, ka šī migrācija nebūs pārāk dārga un laikietilpīga. Ērtākais vaids noskaidrot šis vērtības ir izmantot jau gatavu rīku kas māk izanalizēt veco datu bāzi un dot atbildi cik ilgi un cik dārgi būs migrēt. Taču diemžēl nav tāda rīka kas varētu veikt analīzi Adabas D un Oracle datu bāzes migrācijai. Tātad šo soli patreiz mums ir jāveic pašrocīgi. Tas nozīme, ņemam eksistējošo Oracle datu bāzi un skatāmies vai

1. datu modeli, kas ir patreiz, mēs varēsim bez izmaiņām pārnest uz Adabas D. Kaut arī Adabas D, tāpat kā Oracle, atbalsta relāciju modeļus, ir dažas Oracle iespējas, kas netiek atbalstīti Adabas D datu bāzēs. Kā piemērs var būt objektu tipu un kolekciju uzturēšana. Tas ir ja pašreizējā datu bāzē ir kādi objektu tipi, kuri tiek izmantoti vairākās vietās, tad to būs principiāli jāpārtaisa. Tas prasa papildus laiku, jo ir jāizpēta kāda veida tie tiek izmantoti, jāpārliecinās, ka to vispār būs iespējams aizvietot, izmaiņas jānoprojektē un, protams, jāuztaisa. Savādāk, ja objektu tipi netika izmantoti Oracle datu bāze, tātad datu modeļa pārnešanai nevajadzētu būt pārāk sarežģītai, jo pārējais principā Oracle un Adabas D datu bāzēm ir līdzīgs.
2. tabulas var tikt pārnestas uz Oracle datu bāzi ar visam primāram un ārējam atslēgām, ierobežojumiem, datu tipiem, indeksiem, komentāriem. Principā visi datu tipi Oracle un Adabas D ir vienlīdzīgi, taču Oracle atbalsta arī tādus datu tipus, kas nav atbalstīti Adabas D datu bāzēs. Kā piemērs varētu būt CLOB datu tips. Adabas D principā ir līdzīgs datu tips LONG, taču ja Adabas D tabulā ir kolonas ar tipu LONG, tad tajās var

ielikt lielākais 2.1 GB, taču Oracle tabulas kolonā ar tipu CLOB vai BLOB var ielikt līdz 4 GB. Un tās pats ir ar FIXED tipu, kas principā ir līdzīgs Oracle NUMBER datu tipam, taču FIXED atļauj maksimāli 18 ciparus, taču NUMBER atļauj 38 ciparus. Tāpēc ir vērts padomāt vai datu tipu atšķirības netraucēs programmatūras darbībai. Protams, ja netiek paredzēta sadarbība ar XML un LOB datu tipu uzturēšana, tad laikam šis atšķirības nebūs tik jūtamas. Vēl ir problēma ar UNICODE uzturēšanu. Patreizēja Adabas D 13.01 versija neatbalsta UNICODE, tikai ASCII un EBCDIC, kas nozīme kā visi latviešu burti pazudis Adabas D datu bāzē. Vēl Adabas D neatbalsta tabulu sadalīšanu vairākos nodalījumiem (partitions), tātad ja tabula ir liela, tad vaicājumi uz tas pildīsies ilgāk nekā agrāk Oracle datu bāzē.

Tātad, ja eksistējoša datu bāze nesatur LOB datu tipus vai 2GB ir pietiekams apjoms datiem LOB kolonās, ja projekts ir angļu valodā un ja visi pārēji datu tipi var būt pārnesti uz līdzīgiem ar īsākiem atļautiem garumiem nesabojājot datus, tad principā var domāt par tālāko datu bāzes migrāciju.

3. Adabas D SQL vaicājumi atšķiras no Oracle vaicājumiem un kāda apmērā (gan parastie, gan dinamiskie). Tā kā Oracle atbalsta SQL3 standartu un Adabas D tikai SQL2, tad noteikti šeit būs daudz kas atšķirīgs. Pats pirmais un nepatīkams moments ir datu bāzes veiktspēja. Oracle veiktspējas uzlabošanai ir paredzēti speciālie norādījumi (hints) un, protams, Adabas D nav nekādu norādījumu. Tāpēc, lai Oracle optimizēts vaicājums strādātu tikpat ātri Adabas D, to būs jāoptimizē maksimāli labi. Tāpēc uz vaicājumu pārnešanu un pēc tam optimizēšanu arī aizies diezgan daudz laika.
4. SQL-PL valodas moduli ir viennozīmīgi pārnesami uz PL/SQL. Šis punkts noteikti būs vissarežģītākais, jo Adabas D SQL-PL un Oracle PL/SQL diezgan stipri atšķiras. Pie kam, ja no Oracle funkcijām bija iespējams izsaukt procedūras, tad Adabas D to neatļauj, tāpēc būs nepieciešams ne

tikai pārveidot gatavas procedūras un funkcijas, bet arī jāuzmanās, lai to, kas ir iekš pārveidotā procedūrā vai funkcijā, ir iespējams palaist uz Adabas D. Vēl Oracle trigeri var būt izsaukti pirms DML operācijām, taču Adabas D ļauj tos izsaukt tikai pēc tiem. Pluss trigeris Adabas D var būt piesaistīts tikai vienai tabulai vai skatījumam, kas ir uz vienas tabulas, nevis skatījumam, kas ir uz vairākām tabulām, vai shēmai vai pat datu bāzei.

Tātad sanāk, ja datu bāze ir neliela (pati Adabas D datu bāze nevar pārsniegt 8 TB), ja nav izmantoti LOB tipi, ja dati tabulās glabājas angļu valodā un ja projekta lietojumprogrammatūra nav uztaisīta uz Oracle formām, ja visi trigeri piesaistīti tikai vienai tabulai, tad iespējams ir jēga migrēt no Oracle uz Adabas D, jo tas būs lētāk nekā uzturēt Oracle datu bāzi, neizmantojot projektā visas Oracle iespējas.

5.2.2 Shēmas un datu migrācija

Kā jau bija teikts, lielākam skaitam no Oracle datu tipiem eksistē līdzīgi datu tipi Adabas D datu bāzē. Tāpat Adabas D atbalsta primāras un ārējas atslēgas, pārbaudes ierobežojumus, indeksus, DEFAULT laukus, NOT NULL laukus, tabulas un kolonas komentāru veidošanu. Tātad, lai izveidot jauno datu bāzes modeli, ir jāizeksportē CREATE priekšrakstus, jāpiemodificē un tad tos var palaist uz jaunizveidotas Adabas D datu bāzes.

Oracle un Adabas D tabulas īpašību neatbilstības:

- Tabulas un kolonu nosaukumi nevar būt garāki par 18 simboliem. Tas nozīme, ka visus garus tabulu nosaukumus nāksies saīsināt. Un līdz ar to šos nosaukumus nāksies mainīt visās vietās, kur tie tiek izmantoti.
- Oracle ir vairāki datu tipi, kas nav atbalstīti Adabas D datu bāzēs. Kā piemērs, LOB datu tipi (BFILE, BLOB, CLOB, NCLOB), XMLType, TIMESTAMP WITH TIME ZONE, INTERVAL DAY TO SECOND, ROWID, UNROWID. Protams, var mēģināt saglabāt LOB datu tipu, pārvēršot to par LONG un zaudējot 2 GB datu informācijas (maksimāli LONG atļauj 2 GB, bet LOB tipi – 4 GB). XMLType arī var pārtaisīt LONG datu tipā, tāpat

zaudējot 2 GB datu informācijas. TIMESTAMP WITH TIME ZONE var pārtaisīt par parasto TIMESTAMP, pazaudējot informāciju par laika zonu. Var mēģināt pārvērst TIMESTAMP WITH TIME ZONE un INTERVAL DAY TO SECOND datu tipus uz VARCHAR un pēc tam strādāt ar tiem kā ar virknēm, taču ir jāuzmanās no datuma formāta. Adabas D nav funkcijas TO_DATE un vajadzīgā datu formātā mēs nevarēsim dabūt datumu, tāpēc labāk uzreiz pārveidot šos datu tipus formātā YYYYMMDDHHMMSSmmmμμμ. Taču ROWID un UNROWID datu tipiem nav līdzīgu datu tipu Adabas D datu bāzē, tāpēc nāksies tos pārtaisīt par CHAR datu tipiem.

- Adabas D neatbalsta sekvences kā atsevišķus datu bāzes objektus, bet ir speciāls datu tips, kas principā ir tas pats. Tāpēc tas tabulu kolonas, kas agrāk bija ar sekvencēm, nāksies pārnest uz Adabas D ar datu tipu SERIAL.
- Adabas D neatbalsta tabulu sadalīšanu vairākos nodalījumos. Tāpēc tos būs jānovāc no tabulu CREATE teikumiem.
- Ja Oracle bija kāda datu bāzes objekta nosaukums, kas ir rezervēts vārds Adabas D datu bāzē, tad to labāk pārtaisīt par ierobežotu identifikatoru. Vispār būtu labi visus Oracle identifikatorus pārvērst par ierobežotiem, t.i. atdalot tos ar ″ zīmēm.
- Adabas D nav tādas iespējas norādīt tabulvietu, jo Adabas D pats automātiski izkārto un sadala brīvo atmiņu uz datu bāzes objektiem. Tāpēc tagad nav vajadzīgs norādīt cik daudz vietas ir jāizdala uz tabulām, atslēgām un indeksiem.
- Tā kā Adabas D unikalitātes ierobežojumi nevar būt definēti pēc tabulu izveidošanas, tad tos ir jādefinē kopā ar CREATE TABLE teikumu. Būs labāk, ja tur pat būs arī primāras atslēgas un pārbaudes ierobežojumi.
- Adabas D un Oracle aizvietojamie datu tipi

Oracle	Adabas D
`VARCHAR2(N) [BYTE | CHAR]`, kur N <= 4000	`VARCHAR(N)`, kur N <= 4000

NVARCHAR2(size), kur N <= 4000	-
NUMBER(p,s), kur maksimāli var būt 38 cipari	FIXED(N,M), kur maksimāli var būt 18 cipari
LONG, kur var būt maksimāli 2 GB	LONG, kur maksimāli var būt 2.1 GB
DATE, kur noklusēts formāts ir YYYY-MM-DD	DATE, kur noklusēts formāts ir YYYYMMDD
TIMESTAMP(N), kur noklusēts formāts ir YYYY-MM-DD HH:MM:SS.mmm	TIMESTAMP, kur noklusēts formāts ir YYYYMMDDHHMMSSmmmμμμ
TIMESTAMP(N) WITH TIME ZONE	-
TIMESTAMP(N) WITH LOCAL TIME ZONE	-
INTERVAL YEAR(year_precision) TO MONTH	-
INTERVAL DAY(day_precision) TO SECOND(fractional_seconds_precision)	-
RAW(N), kur N <= 2000	BYTE
LONG RAW, kur var būt maksimāli 2 GB	LONG, kur maksimāli var būt 2.1 GB
ROWID	-
UROWID [(size)]	-
CHAR(N)[BYTE | CHAR], kur N <= 2000	CHAR (N), kur N <= 4000
NCHAR(N), kur N <= 2000	-
CLOB, kur var būt maksimāli 4 GB	LONG, kur maksimāli var būt 2.1 GB
NCLOB, kur var būt maksimāli 4 GB	LONG, kur maksimāli var būt 2.1 GB
BLOB, kur var būt maksimāli 4 GB	LONG, kur maksimāli var būt 2.1 GB
BFILE, kur var būt maksimāli 4 GB	LONG, kur maksimāli var būt 2.1 GB
XMLType, kur var būt maksimāli 4 GB	LONG, kur maksimāli var būt 2.1 GB

Tabula 10 – Oracle un Adabas D aizvietojamie datu tipi

Oracle speciali datu bāzes eksportam un importam ir uztaisītas eksporta un importa utilītprogrammas (utilities). Tātad caur komandas rindu palaižot komandu exp ar vajadzīgiem mums parametriem, var dabūt Oracle datu bāzes shēmas instalēšanas skriptus. Un protams, eksportētie faili satur instalācijas skriptus atbilstošus Oracle datu bāzei, tātad tos būs jāpārtaisa uz Adabas D formātu.

Piemērs tabulas veidošanas skriptam, ko uzģenerēja eksporta utilītprogramma:

```
CREATE TABLE "CUSTOMER"
("CNO" NUMBER(4, 0),
"TITLE" VARCHAR2(7),
"FIRSTNAME" VARCHAR2(10),
"NAME" VARCHAR2(10) NOT NULL ENABLE,
"ZIP" VARCHAR2(5),
"ADDRESS" VARCHAR2(25) NOT NULL ENABLE,
"PHONENO" VARCHAR2(15)
)
PCTFREE 10
PCTUSED 40
INITRANS 1
MAXTRANS 255
STORAGE(INITIAL 65536
FREELISTS 1
FREELIST
GROUPS 1)
TABLESPACE "SYSTEM" LOGGING NOCOMPRESS
/

ALTER TABLE "CUSTOMER" ADD  PRIMARY KEY ("CNO")
USING INDEX PCTFREE 10
INITRANS 2
MAXTRANS 255
STORAGE(
INITIAL 65536
FREELISTS 1
FREELIST
GROUPS 1)
TABLESPACE "SYSTEM" LOGGING ENABLE
/

ALTER TABLE "CUSTOMER" ADD
CONSTRAINT "TITLE"
CHECK (TITLE IN ('Mr', 'Mrs', 'Company')) ENABLE NOVALIDATE
/

ALTER TABLE "CUSTOMER" ADD  CONSTRAINT "CUST_UNQ1" UNIQUE ("TITLE",
"FIRSTNAME", "NAME")
USING INDEX PCTFREE 10
INITRANS 2
MAXTRANS 255
STORAGE(
INITIAL 65536
FREELISTS 1
FREELIST GROUPS 1)
TABLESPACE "SYSTEM" LOGGING ENABLE
/
```

Lai būtu iespējams palaist šo skriptu Adabas D, to ir jāpārveido šādi:

```
CREATE TABLE "CUSTOMER"
("CNO" FIXED(4,0),
"TITLE" VARCHAR2(7),
"FIRSTNAME" VARCHAR2(10),
"NAME" VARCHAR2(10) NOT NULL,
"ZIP" VARCHAR2(5),
```

```
"ADDRESS" VARCHAR2(25) NOT NULL,
"PHONENO" VARCHAR2(15),
CONSTRAINT "TITLE" CHECK TITLE IN ('Mr', 'Mrs', 'Company'),
CONSTRAINT "CUST_UNQ1" UNIQUE ("TITLE", "FIRSTNAME", "NAME"),
PRIMARY KEY ("CNO")
)
/ *
```

Izņemot skriptu pārveidošanu vajadzētu izanalizēt tabulu references un uztaisīt atsevišķus failiņus, kuros būs atsevišķi loģiski sakārtoti tabulu, sinonīmu, GRANT, ārējo atslēgu, indeksu un skatījumu veidošanas skripti, kā arī datu ielādēšanas skripti, jo tā būs vieglāk un ērtāk pārbaudīt vai visi skripti ir pareizi pārtaisīti.

Kad visi skripti ir gatavi, tos var viegli palaist Adabas D datu bāzē izmantojot rīku LOAD.

Instalēšanas secība:

1. Tabulas
2. Sinonīmi
3. Datu ielikšana
4. Indeksi
5. GRANT
6. Skatījumi
7. Datu bāzes saites

5.2.3 Biznesa loģikas migrācija

Šajā fāzē vajadzēs

- izeksportēt visus PL/SQL moduļus;
- izanalizēt PL/SQL un SQL-PL valodas un iespēju atšķirības;
- pārtaisīt visus PL/SQL moduļus uz SQL-PL moduļiem;
- uzinstalēt gatavus SQL-PL moduļus Adabas D datu bāzē.

Kad visas DB procedūras, DB funkcijas, procedūras, funkcijas un trigeri ir izeksportēti, sakas to analīze. Ir jāsalīdzina SQL-PL un PL/SQL valodu struktūras un to atšķirības, kā arī jāzina kā var pārveidot konkrētas struktūras.

Zemāk ir aprakstīts kā jāizskatās Adabas D SQL-PL programmām Tabula 11

Oracle	Adabas D
Bāzes elementi	
Komentāri	

`/*liels komentārs*/`
vai
`--komentārs`
	`/* liels`
	`/* komentārs`

Nosaukumi	
Adabas D jebkurš nosaukums nevar pārsniegt 18 zīmju garumu, tāpēc ja ir kāds Oracle moduļu, mainīgu vai datu bāzes objektu nosaukumus, kas ir garāks par 18 zīmēm, to ir jāsaīsina un jāizlabo visas vietas, kur šis nosaukums parādījās. Oracle maksimālais identifikatoru garums var būt 30 simboli. Ja kāds Oracle identifikators ir Adabas D rezervētais vārds (skatīt Tabula 6 un Tabula 7), tad to ir jāiekļauj apostrofos un tas būs ierobežotais identifikators. Piemērs:	
`PROCEDURE modify (a NUMBER) AS ...`	`PROC ” customer”.”modify” PARMS (a) ...`
Literāli	
Numuru un simbolu literāli sakrīt Oracle un Adabas D.	
Mainīgie	
Oracle mainīgo deklarēšana notiek uzreiz pēc atslēgvārda DECLARE, t.i. anonīmā bloka sakumā (DECLARE ... BEGIN ... END;). Katram mainīgam ir piešķirts savs datu tips un ja vajag noklusētas vērtības. Adabas D DB procedūrās, DB funkcijās, procedūrās un funkcijās nav jānorāda mainīga tipi un pat nav obligāti, ka mainīgais būs deklarēts konkrētajā modulī. Ja mainīgais nav deklarēts SQL-PL modulī un tas priekšā nesatur @ vai @@ zīmes, tad tas ir globālais mainīgais, kuru var izmantot visas procedūras, kas ir iekš programmas. DB procedūrās un funkcijas globālie mainīgie nav atļauti. Tātad, ja mums ir PL/SQL procedūra, kurā pēc DECLARE ir definēti procedūras	

mainīgie, tad tos arī ir jāpārnes uz SQL-PL procedūru, pie kam tos visus ir jāpārskaita vienā rindā atdalot ar komatiem un sakumā pieliekot atslēgas vārdu VAR un tālāk konkrētajā procedūrā katru reizi, kad tāds mainīgais ir izmantot, tam sakumā ir jāpieliek @ zīme, kas liecina, ka šis mainīgais ir lokālais dinamiskais, t.i. pēc procedūras atkārtotas izsaukšanas vecā vērtība nepaliks. Principā lokālus mainīgus var arī nedeklarēt, taču @ zīme ir obligāti jāizmanto.

Ja mums ir pakotne, kuras galvā ir deklarēti mainīgie, tad to deklarēšanu pat nevajag pārnest uz SQL-PL, jo SQL-PL globālus mainīgus nedeklarē. Vienīgais ir jāuzmanās no tā, ka Oracle mainīgu, kas ir deklarēts vienā pakotnes galvā, var izmantot arī citās pakotnes. Šajā gadījumā būs problēmas, jo globālie mainīgie eksistē vienas programmas ietvaros, t.i. globālais mainīgais no vienas programmas jau nebūs tas pats mainīgais, kas ir otrā programmā, pat ja nosaukumi tiem ir vienādi. Protams, šādā gadījumā var apvienot saistītas ar „globāliem" mainīgiem pakotnes, tikai atkal ir jāuzmanās, lai pārējo principā atšķirīgo „globālo" mainīgo nosaukumi nesakristu.

SQL-PL funkcijās nav atļauts izmantot globālus mainīgus, tāpēc tos būs jāpadod kā parametrus.

SQL-PL visi padotie parametri arī skaitās par lokāliem-dinamiskiem mainīgiem (ja tikai tas nav IN parametrs). Tāpēc to izmantošanai SQL-PL modulī arī būs jāliek @ zīme priekšā.

SQL teikumos mainīgiem priekšā pieliek : zīmi.

Un vēl mainīgo nosaukumi, tāpat kā identifikatori, nevar būt garāki par 18 simboliem, kaut gan Oracle atbalsta 30 garus nosaukumus.

```
--procedūrā
DECLARE
  a VARCHAR2(255);
  b NUMBER;
  c DATE;
```

```
VAR a, b, c;
...
@a := 'asdad';
@b := 12;
@c := DATEDIFF(max_departure,
```

BEGIN a := 'asdad'; b := 12; c := max_departure - min_arrival;	min_arrival);
SELECT max_free **INTO** max_free **FROM** sqltravel10.room **WHERE** cno = 1;	SQL (SELECT max_free INTO :@max_free FROM sqltravel10.room WHERE cno = 1);
Izteiksmes	
Aritmētiskie, virknes un loģiskie izteikumi Oracle un Adabas D lielākoties sakrīt. Atšķirības: o SQL-PL aritmētiskās izteiksmēs var izmantot DB funkcijas, tātad, ja mums ir definēta DB funkcija (Oracle funkcija) un mēs to vēlamies izmantot, tad priekšā ir jāieliek % zīme, piemēram, `a := %vsum(b,c,d);` o virknes izteikumos konkatenācijas zīmi || ir jāaizvieto ar & zīmi o loģiskos izteikumos, kad tiek izmantota salīdzināšana (`LIKE 'S%'`), % zīmes ir jāaizvieto ar * zīmēm o ja PL/SQL tika izmantots ** operators, tad to ir jāaizvieto ar DB funkciju, kas mācēs kāpināt skaitļus norādītajā pakāpē	
Kontroles struktūras	
IF izteikums	

IF <condition1> THEN
  <sequence_of_statements1>
[ELSIF <condition2> THEN
  <sequence_of_statements2>]
[ELSE
  <sequence_of_statements3>]
END IF;
	IF <condition1> THEN
	<sequence_of_statements1>
	[ELSE
	  [IF THEN
	  <sequence_of_statements2>
	  [ELSE <sequence_of_statements3>]]
	<sequence_of_statements4>]
	END IF;

CASE izteikums	
[<<label_name>>] CASE selector WHEN expression1 THEN sequence_of_statements1; WHEN expression2 THEN sequence_of_statements2; ... WHEN expressionN THEN sequence_of_statementsN; [ELSE sequence_of_statementsN+1;] END CASE [label_name]; vai [<<label_name>>] CASE WHEN search_condition1 THEN sequence_of_statements1; WHEN search_condition2 THEN	CASE <expr> OF <value spec list> : <compound> ; [<value spec list> : <compound> ;] [OTHERWISE <compound>] END;

`sequence_of_statements2;`
`...`
`WHEN search_conditionN THEN`
`sequence_of_statementsN;`
`[ELSE sequence_of_statementsN+1;]`
`END CASE [label_name];`	

LOOP izteikums	

`LOOP`
`  <sequence_of_statements>`
`  EXIT;`
`END LOOP;`
	Tādas pašas konstrukcijas Adabas D nav, tāpēc var mēģināt to aizstāt ar REPEAT vai WHILE izteikumiem. Līdzīgas EXIT komandai Adabas D arī nav.

WHILE-LOOP izteikums	

`WHILE <boolean expr> LOOP`
`  <stmt>;`
`  ...`
`END LOOP;`
	`WHILE <boolean expr> DO <compound>`

FOR-LOOP izteikums	

`FOR <variable> IN [REVERSE]`
`  <expr>..<expr> LOOP`
`  <compound>`
`END LOOP;`
vai
`FOR <variable> IN [REVERSE]`
`  (<select_stmt>) LOOP`
`  <compound>`
`END LOOP;`
	`FOR <variable> := <expr> TO <expr>`
	`DO <compound>`
	vai
	`FOR <variable> := <expr> DOWNTO`
	`<expr> DO <compound>`

GOTO izteikums	

`GOTO label;`
`  ...`
`<<label>>`
`statement;`
	`SKIP label;`
	`  ...`
	`label : statement;`

RETURN izteikums	
Šis izteikums sakrīt gan Adabas D, gan Oracle. To izmanto lielākoties funkcijās, taču var arī procedūrās.	
NULL izteikums	
NULL izteikums nedara neko un ļauj pariet pie nākamas komandas. Adabas D tādu izteikumu neatbalsta, tāpēc to var aizvietot ar kaut ko bezjēdzīgo un kas vienmēr būs patiess, piemēram, `SQL (SELECT 1 FROM dual WHERE 1=1);`	
Moduļu izsaukšana	
Procedūru izsaukumi	
Adabas D procedūras iespējams izsaukt trīs veidos, t.i. CALL, kad pēc izsauktas procedūras pabeigšanas mēs atgriežamies atpakaļ; SWITCH, kad pēc izsauktas	

procedūras pabeigšanas mēs atpakaļ neatgriežamies; un SWITCHCALL, kad pēc izsauktas procedūras pabeigšanas mēs atgriežamies atpakaļ. Starpība starp CALL un SWITCHCALL ir tāda, ka ar CALL mēs varam izsaukt tikai programmai piederošas procedūras, taču SWITCHCALL var izsaukt procedūru no citas programmas. SWITCH arī izsauc procedūru no citas programmas. Procedūras ir iespējams izsaukt no DB procedūrām, procedūrām un trigeriem. No DB funkcijām un funkcijām aizliegts izsaukt procedūras.

Tāpēc ja mums ir PL/SQL pakotne, kas principā ir līdzīgs SQL-PL programmai, un iekšējas procedūras izsauc citas iekšējas procedūras, tad tos izsauksim ar CALL, tomēr ja kāda iekšēja procedūra izsauc procedūru no citas pakotnes, tad lietosim SWITCHCALL. SWITCH mēs nelietosim, jo Oracle nav tādas iespējas aiziet uz citu procedūru un atpakaļ neatgriezties. Ja Oacle no kādas funkcijas bija izsaukta procedūra, tad to procedūru nāksies pārveidot uz funkciju.

`--Pakotnē customer customer.display (cno, resdat);`	`CALL PROC display PARMS (cno, resdat);`
`--Pakotnē customer bills.check_bill;`	`SWITCHCALL bills CALL PROC check_bill;`

Funkciju izsaukumi

Funkcijas var būt izsauktas no DB procedūrām, procedūrām, trigeriem un funkcijām. Funkciju izsaukums Oracle notiek vairākos veidos, t.i. funkciju var piešķirt mainīgajam BEGIN END blokā, funkciju var piešķirt mainīgajam kā nodefinēto vērtību mainīgo deklarēšanas blokā, funkciju var izmantot kā loģisku izteiksmi, piemēram, IF-THEN izteikumā, funkciju var izmantot SQL teikumos, un beidzot funkciju var izmantot padodot procedūrai vai citai funkcijai parametrus. Adabas D nav atļauts piešķirt funkcijas rezultātu deklarēšanas blokā, jo tur vienkārši tiek pārskaitīti lokālie mainīgie, bet no tā var viegli izvairīties, piešķirot mainīgam funkcijas vērtību pēc deklarēšanas bloka. Tāpat nedrīkst padot mainīgus ar funkcijas vērtībām un šo situāciju var atrisināt piešķirot mainīgam funkcijas vērtību pirms procedūras vai funkcijas izsaukuma. Funkciju izsaukumi SQL-PL notiek ar % zīmes palīdzību.

`v_sum := vsum (a, b, c);`	`@v_sum := %vsum (@a, @b, @c);`

DB procedūru izsaukumi

Adabas D DB procedūras izsaukšanas sintakse ir CALL DBPROC. DB

procedūras var būt izsauktas no procedūrām un trigeriem. Vēl tas var būt izsauktas no DB procedūrām, bet ar procedūras izsaukšanas sintaksi, t.i. CALL PROC. Migrējot no Oracle principā visas procedūras vai nu paliks par procedūrām vai par funkcijām. DB procedūras izmantosim tikai „sazināšanai" ar ārējam programmām, ja tādas ir, t.i. visi SQL-PL izsaukumi no citam valodām un procedūras, kas pašas izsauc citas valodas programmas mēs pārveidosim par DB procedūrām. DB procedūru izsaukšanas veidi:

```
CALL DBPROC [[<owner>.]<progname>.]<dbproc name> [PARMS (<param>,...]
[WITH COMMIT]
```

vai

```
SQL ( DBPROC[EDURE] [[<owner>.]<progname>.]<dbproc name> [(<host
var>,...)] [WITH COMMIT] )
```

DB funkciju izsaukumi

DB funkcijas principā ir ļoti primitīvas funkcijas, kas aizvieto kādu citas valodas iebūvēto funkciju. Un tā kā iebūvētas funkcijas parasti izmanto SQL teikumos, tad arī DB funkcijas tiek izsauktas SQL teikumu ietvaros.

```
SELECT NVL(first_name||last_name,
name)
INTO v_full_name
FROM customers
WHERE id = v_id;
```

```
SQL (SELECT
nvl_val(first_name||last_name,
name)
     INTO :@v_full_name
     FROM CUSTOMERS
     WHERE id = :v_id);
```

Trigeru izsaukumi

Oracle trigeri var būt piesieti tabulai, skatījumam, shēmai vai datu bāzei un tie var izsaukties pirms, pēc vai definētas darbības vietā. Adabas D trigerus ļauj piesiet tikai tabulām un skatījumiem, kas ir uz vienas tabulas, un trigeris tiek izsaukts pēc definētām darbībām. Adabas D šīs darbības var būt INDERT, UPDATE vai DELETE uz konkrēto tabulu un tikko šī darbība ir izpildīta palaižas trigeris.

SQL izmantošana

Piekļuve datu bāzei

Oracle PL/SQL obligāti katrs SELECT teikums atgriež rezultātus definētos mainīgos. Adabas D SELCT teikums var uzreiz neatgriezt rezultātus mainīgos, to var izdarīt vēlāk ar FETCH palīdzību. Vēl Adabas D visi SQL vaicājumi apzīmējas ar atslēgvārdu SQL un ir ierakstīti apaļās iekavās (t.i. nedinamiskie SQL vaicājumi), bet Oracle SQL vaicājumi ir bez palīg atslēgvārdiem.

Datu manipulācijas:	
Gan Adabas D, gan Oracle izmanto INSERT, UPDATE, DELETE un SELECT komandas. Toties Oracle vēl ir MERGE.	
INSERT INTO <table name> [(<column name>,...)] VALUES (<extended expression>,...)	SQL (INSERT [INTO] <table name> [(<column name>,...)] VALUES (<extended expression>,...))
INSERT INTO <table name> [(<column name>,...)] <SELECT Statement>	SQL (INSERT [INTO] <table name> [(<column name>,...)] <query expression>)
INSERT ALL INTO <table_name> VALUES <column_name_list) INTO <table_name> VALUES <column_name_list) ... <SELECT Statement>	Jāsadala vairākos INSERTos
INSERT WHEN (<condition>) THEN INTO <table_name> (<column_list>) VALUES (<values_list>) WHEN (<condition>) THEN INTO <table_name> (<column_list>) VALUES (<values_list>) ELSE INTO <table_name> (<column_list>) VALUES (<values_list>) SELECT <column_list> FROM <table_name>	Jāsadala uz SELECT teikumu, un CASE izteiksmi, kurā būs visi šie INSERT teikumi.
UPDATE <table_name> SET <column_name> = <value> WHERE <search condition>	SQL (UPDATE <table name> SET <column name> = <value> WHERE <search condition>)
UPDATE <table_name> <alias> SET (<column_name>,<column_name>,...) = <subquery> [WHERE <search condition>]	Iepriekš ar SELECT atlasīt vajadzīgus datus un pēc tam: SQL (UPDATE <table_name> SET (<column>,...) VALUES (<extended value spec>,...) [WHERE <search condition>])
DELETE FROM <table_name> [WHERE <condition>]	SQL (DELETE [FROM] <table name> [<reference name>] [KEY <key spec>,...] [WHERE <search condition>])
SELECT [DISTINCT | UNIQUE] <select column>, ... INTO <variable>, ... FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]	SQL (SELECT [DISTINCT] <select column>, ... [INTO :<variable>, ...] FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>])
MERGE <hint> INTO <table_name> USING <table_view_or_query> ON (<condition>)	Nav tāda, bet var aizvietot ar INSERT

`WHEN MATCHED THEN <update_clause> WHEN NOT MATCHED THEN <insert_clause>`	un UPDATE komandām.
Adabas D SQL teikumos visiem lokāliem un globāliem mainīgiem priekšā ir jāliek „:" zīme.	
Transakciju kontrole:	
Oracle transakcijas mēs varam paši kontrolēt izmantojot COMMIT vai ROLLBACK komandas. Adabas ar transakcijām viss ir mazliet savādāk... Kad tiek atverta SQL-PL programma, atveras arī jauna transakcija, kuru var apturēt ar COMMIT WORK vai ROLLBACK WORK komandām. Ja mēs norādam procedūras izsaukumā WITH COMMIT, tad ja tā beidzas veiksmīgi tiek uztaisīts COMMIT, savādāk nekas nenotiek. Var arī izsaukt procedūras ar opciju AUTOCOMMIT OFF, tad ne COMMIT ne ROLLBACK netiks uztaisīti, kāmēr tas nebūs iekodēts kodā. Taču Adabas D vēl ļauj izmantot apakštransakcijas, t.i. SUBTRANS BEGIN, SUBTRANS END un SUBTRANS ROLLBACK, kuras var būt pielietotas jebkurā vietā, tas principā ir lidzīgi Oracle SAVEPOINT un COMMIT vai SAVEPOINT un ROLLBACK.	
SQL Operatori	

`ALL`
`ANY,SOME`
`BETWEEN`
`EXISTS`
`IN`
`IS NULL`
`LIKE`
`INTERSECT, INTERSECT ALL`
`MINUS`
`UNION, UNION ALL`
`DISTINCT`
`PRIOR`
`AND, OR, NOT`
	`ALL`
	`ANY,SOME`
	`BETWEEN`
	`EXICTS`
	`IN`
	`IS NULL`
	`LIKE`
	`INTERSECT, INTERSECT ALL`
	`MINUS, EXCEPT`
	`UNION, UNION ALL`
	`DISTINCT`
	`-`
	`AND, OR, NOT`

Kursori	
Kursori Oracle PL/SQL var būt atsevišķi deklarēti vai tos izmanto iekš FOR ciklā. Adabas D kursoru var tikai definēt atsevišķi.	
`CURSOR <result table name> [(<parameter>,...)] IS <select statement>;`	`DECLARE <result table name> CURSOR FOR <select statement>;`
`FOR <variable> IN [REVERSE] (<select_stmt>) LOOP <compound> END LOOP;`	1. deklarējam kursoru; 2. `OPEN <result table name>` 3. WHILE ciklā eajm pa kursoru, kamēr $RC <> 100. 4. `CLOSE [<result table name>]`

Kolekcijas	
Oracle ir trīs kolekciju veidi, t.i. asociatīvie masīvi, mainīga lieluma masīvi un ligzdošanas tabulas. Kolekcijas var saglabāt datu bāzē, ja vienu un to pašu kolekciju izmanto vairāk nekā vienā vietā. Taču tas var arī definēt pakotnēs, procedūrās un funkcijās kā atsevišķu datu tipu. Tā kā Adabas D deklarēšanas blokā neuzrada datu tipus, tad arī kolekcijas netiek uzdefinētas iepriekš, tas definējas laikā kad tajās ieliek vērtības.	

`TYPE type_name IS TABLE OF element_type [NOT NULL];`
 	
`TYPE type_name IS {VARRAY | VARYING ARRAY} (size_limit) OF element_type [NOT NULL];`
 	
`TYPE type_name IS TABLE OF element_type [NOT NULL] INDEX BY [PLS_INTEGER | BINARY_INTEGER | VARCHAR2(size_limit)];`
	Nedeklarē

`SELECT [DISTINCT] BULK COLLECT INTO <select column>,... FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]`	`SELECT [DISTINCT] <result table name> (<select column>,...) FROM <table name> [WHERE <search condition>] [GROUP BY <expression>, ...] [HAVING <search condition>] [ORDER <select column>]`
Dinamisko SQL izmantošana	
Dinamiskā SQL izteiksme tiek veidota no SQL izteiksmēm, izmantojot atdalošo izteiksmi un mainīgus. Tā var ērti piekļūt datu bāzei ar vajadzīgo mums SQL izteiskmi.	

`EXECUTE IMMEDIATE <expr>`	`SQL [<dbname>] EXECUTE IMMEDIATE <expr>`
	
	`SQL EXECUTE IMMEDIATE <expr> USING <expr list>`

Kļūdu apstrāde	
Oracle par kļūdām tiek uzskatīti izņēmumi. Lai dabūt un apstrādāt izņēmumu visi SQL vaicājumi tiek ielikti	Pēc katra SQL vaicājuma vai SQL-PL moduļa izpildes tiek atgriezta vērtība $RC un $RT parametros. Un ja šī

tā saucama anonīmā blokā un pēc izteikuma aplamas izpildes tiek apstrādāti izņēmumi. Oracle ir definētie izņēmumi, bet tos var definēt arī PL/SQL modulī.	vērtība nav vienāda ar 0, tad tas nozīme, ka notika kļūda. Pēc $RC (kļūdas kods) var noskaidrot kas tā bija par kļūdu. $RT parametrs satur kļūdas paskaidrojumu angļu valodā. Vēl Adabas D ir iespējams kļūdas apstrādāt TRY-CATCH blokā, bet tas ir vairāk paredzēts procedūru un funkciju izsaukumiem.

`DECLARE`
`  my_exception EXCEPTION;`
`...`
`BEGIN`
`  SELECT 1`
`  INTO v_tmp`
`  FROM customers`
`  WHERE status = 'ANNULED'`
`  AND bill_sum > 0;`
 	
`  RAISE my_exception;`
 	
`EXCEPTION`
`  WHEN my_exception THEN`
`  dbms_output.put_line('Anuletam klientam palika parads!');`
`  WHEN OTHERS THEN`
`  dbms_output.put_line(SQLERRM);`
`END;`
	`SQL (SELECT 1`
	`  FROM customers`
	`  WHERE status = 'ANNULED'`
	`  AND bill_sum > 0);`
	`IF $RC = 0`
	`  THEN MESSAGE := 'Anuletam klientam palika parads!'`
	`  ELSE MESSAGE := 'Error '& $RC;`

	`TRY`
	`  BEGIN`
	`  CALL PROC do_command (...);`
	`  END`
	`CATCH errno OF`
	`  16102 : ERROR := 'There is not enough memory available for this command';`
	`  16801 : ERROR := 'Command interrupted';`
	`END;`

SQL-PL moduļu pārveidošana	
Oracle atbalsta sekojošus moduļus: pakotnes, procedūras, funkcijas, trigerus un objektu tipus. Adabas D atbalsta: DB procedūras, DB funkcijas, procedūras, funkcijas un trigerus. Tātad ja mūsu datu bāzē bija objektu tipi, mēs nevarēsim tos uzturēt Adabas D.	

Pirms nolemt par ko pārvērst PL/SQL procedūru ir jāzina vai šo procedūru izsauc no ārējas programmas, ja jā, tad procedūru pārvēršam par DB procedūru. Vai šī procedūra izsauc citas ne Oracle programmas, ja jā, tad arī šādu procedūru pārvēršam par DB procedūru. Vai šī procedūra izsauc procedūras no citam pakotnēm, ja izsauc, tad to ir jāpārvērš par procedūru. Ja kāda procedūra atbilst gan DB procedūras, gan procedūras kriteriem, tad to pārveidojam par procedūru un ir jāveido papildus DB procedūra, kura izsauks mūsu procedūru.

Pirms nolēmt par ko pārvērst PL/SQL funkciju ir jāzina vai šī funkcija izsauc kādas procedūras, ja jā, tad Adabas D tā nedrīkst būt un tāpēc to vajag pārvērst par procedūru, kura caur saviem parametriem atgriezis bijušas funkcijas rezultātu. Ja funkcija nesatur nevienu SQL izteiksmes un neizsauc neko citu, tad to pat labāk pārvērst par DB funkciju. Ja tomēr funkcijā ir kādi SQL izteiksmes un nav izsauktas procedūras, tad to pārvēršam par funkciju.

Ar SQL-PL trigeriem nav viss tik vienkārši. Pirmkārt, viss kās seko PL/SQL trigerī pēc atslēgvārda OF (vai ja tur pat nav tāda atslēgvārda) būs jāieraksta SQL-PL trigera parametros ar tieši tādiem nosaukumiem un dati tipiem kādi tie ir tabulā. Otrkārt, trigeri mēs uzreiz nevarēsim piesiet tabulai, tas būs jādara SQL-PL rīkā katram trigerim atsevišķi, kur norādīsim uz kādam darbībām tam ir jāreaģē un uz kādas tabulas (kolonam) šis trigeris atradīsies. Treškārt, SQL-PL trigeris nevar būt uz datu bāzes, shēmas un skatījumiem, kas ir vairāk nekā uz 1 tabulas bāzēti, kā tas bija Oracle. Tāpēc tos trigerus nāksies aizvietot ar papildus kodu visās tas vietas, kur mainās tas objekts, uz kura agrāk bija Oracle trigeris.

Un vēl Oracle ir ļoti daudz iebūvēto funkciju, kas nav atbalstītas Adabas D, tāpēc vismaz tas, kas ir izmantotas mūsu PL/SQL moduļos arī būs jāuztaisa. Pie tādam attiecas NVL, COALESCE, DEPTH, ADD_MONTH, INSTR, LAST_DAY, MONGTH_BETWEEN, u.t.t.

Pakotnes pārnest ir visvieglāk, vienīgais kas no tam paliek ir nosaukums un

mainīgie, kas SQL-PL moduļos skaitīsies kā globālie. Vienīgais vajag uzmanīties ar šiem mainīgajiem, jo pat ar vienu un to pašu nosaukumu mainīgie no dažādam programmām ir pavisam citi mainīgie. Tāpēc ja mēs izsaucam vienu „globālu" mainīgu no vienas pakotnes citā, tad labāk tas pakotnes apvienot vienā programmā.

Un, protams, nedrīkst aizmirst, ka identifikatori Adabas D nedrīkst būt garāki par 18 zīmēm, tas nozīme, kā būs jāsaīsina gan moduļu nosaukumi, gan jāizlabo jau izlaboto tabulu un skatījumu nosaukumi.

Oracle iebūvēto funkciju piemērs:

```
a := NVL(b,c)
```

```
DBFUNC DBFUN.nvl PARMS (IN var1
VARCHAR(1000), IN var2
VARCHAR(1000)) : VARCHAR(1000);

VAR resault_name;

IF var1 IS NOT NULL THEN
  resault_name := var1;
ELSE
  resault_name := var2;

RETURN(resault_name);
ENDMODULE
```

Instalēšanai katra SQL-PL moduļa beigās ir jāpieliek ENDMODULE atslēgvārdu.

Piemērs trigera pārveidošanai:

```
CREATE OR REPLACE TRIGGER cust_full_name
AFTER INSERT OR UPDATE OF cust_group, name, first_name, last_name
ON customer
REFERENCING NEW AS NEW OLD AS OLD
BEGIN
  IF :new.cust_group = 'J' THEN
    IF :new.NAME IS NOT NULL THEN
      :new.full_name := :new.NAME;
    END IF;

  ELSIF :new.cust_group = 'F' THEN
    IF :new.first_name IS NOT NULL THEN
      :new.full_name := :new.first_name;
      IF :new.last_name IS NOT NULL THEN
        :new.full_name := :new.full_name || ' ' || :new.last_name;
      END IF;
    ELSIF :new.last_name IS NOT NULL THEN
      :new.full_name := :new.last_name;
    END IF;
```

```
  END IF;
END;
/
```

```
TRIGGER triggs20.cust_full_name( IN NEW.cust_group CHAR(1),
                                 IN NEW.name VARCHAR(60),
                                 IN NEW.first_name VARCHAR(30),
                                 IN NEW.last_name VARCHAR(30))

IF :new.cust_group = 'J' THEN
  IF :new.NAME IS NOT NULL THEN
    :new.full_name := :new.NAME;
  END IF;

ELSIF :new.cust_group = 'F' THEN
  IF :new.first_name IS NOT NULL THEN
    :new.full_name := :new.first_name;
    IF :new.last_name IS NOT NULL THEN
      :new.full_name := :new.full_name || ' ' || :new.last_name;
    END IF;
  ELSIF :new.last_name IS NOT NULL THEN
    :new.full_name := :new.last_name;
  END IF;

END IF;
ENDMODULE
```

Kad trigeris ir uzinstalēts, ejam uz Adabas D SQL-PL rīku un ,izmantojot Selection -> Create IN DB funkciju, izveidojam trigeri datu bāzē, kur kā parametrus norādām trigera nosaukumu, uz kādas tabulas atradīsies šis trigeris, norādām kolonas ar kuram tas ir saistīts, ka arī izvēlāmies uz kādu darbību trigerim ir jāreaģē (INSERT, UPDATE, DELETE).

Tabula 11 - Oracle PL/SQL un Adabas D SQL-PL valodu salīdzinājums, iespējami risinājumi

Instalēšanas secība:

1. DB funkcijas
2. Procedūras
3. Funkcijas
4. Trigeri
5. DB procedūras

Lai DB procedūras un DB funkcijas uzliktos datu bāzē vajag caur SQL-PL rīku iziet cauri katrai un izveidot tas datu bāzē ar komandu Selection -> Create in DB. Tikko DB procedūras un DB funkcijas tiks izveidotas datu bāzē, tad visas saistītas ar tiem funkcijas un procedūras arī ielieksies datu bāzē.

5.2.4 Iegūtas datu bāzes testēšana

Tikko datu bāze ir pārnesta un viss ir uzinstalēts, tad ir jātestē vai tiešam viss strādā, tā kā tas bija agrāk. Vislabākais variants būtu izstrādāt programmas, kas pieslēdzas abām datu bāzēm un vienlaicīgi vai pēc kārta izpilda vienas un tas pašas darbības, t.i. palaiž procedūras un funkcijas, ievieto, atjauno vai dzēš vienādus datus tabulās. Un tikko darbība ir izpildīta, tad salīdzina rezultātus. Galvenais lai rezultāts, kuru mēs dabūjam vecā datu bāze, ir tāds pats kā jaunā.

5.2.5 Lietojumprogrammatūras migrācija

Lietojumprogrammatūras ar datu bāzes atbalstu var būt divos veidos, t.i. uzprogrammētas ar citu programmēšanas valodu vai uz datu bāzes valodas pamatā. Pirmais variants ir tad, kad datu bāzes ir vajadzīga tīri datu manipulācijām un parēja projekta daļa ir uzprogrammēta atsevišķi. Kā piemērs ir interneta veikals, labs dizains un visa informācija par precēm glabājas datu bāzē. Šajā variantā migrāciju veikt ir vieglāk, jo principā var viegli noskaidrot kādas vietās notiek sadarbība ar datu bāzi, un ja vajag pielabot tas. Tas ir, mūsu uzdevums ir atrast vai nu uzreiz SQL vaicājumus datu bāzei, vai PL/SQL programmas izsaukumus un izlabot uz Adabas D atbilstošiem vaicājumiem un jau izveidoto SQL-PL moduļu izsaukumiem. Adabas D SQL-PL moduļu izsaukumi notiek pavisam savādāk nekā Oracle PL/SQL izsaukumi, kā arī kļūdu apstrāde notiek savādāk. Tāpēc būs vairāk jāpiestrādā pie SQL-PL moduļu izsaukumiem un sekojošam darbībām.

Ja projekts bija izveidots ar Oracle formu palīdzību, tad tos nāksies migrēt pilnība visus uz kādu citu programmēšanas valodu kas izmanto darbībai ar datu bāzi ODBC vai JDBC draiverus. Visvieglāk ir migrēt no Oracle formām uz Java, un tas ir tāpēc, ka ir jau izstrādāti rīki, kas spēj migrēt no Oracle formām uz J2EE un Oracle Application Development Framework (ADF). Taču šie rīki vienkārši pārveido formas uz Java lietojumprogrammatūru, bet tiek uzskatīts, ka paliks Oracle datu bāze, tāpēc kad formas ir pārveidotas, būs jāatrod visas vietas, kur notiek sadarbība ar datu bāzi un ja vajag jāizlabo tas uz Adabas D datu bāzes formātu.

5.2.6 Testēšana, integrēšana un izvietošana

Sakuma visu ir jāpalaiž, t.i. jāpārliecinās vai mums ir pareizs savienojums ar jaunu datu bāzi, vai kādi no izmainītiem (jaunizveidotiem) moduļiem nepalika nederīgi (invalid). Pēc tam tāpat kā iepriekš pārbaudām vai visi lietojumprogrammatūru moduļi strādā tāpat kā iepriekš, vai iepriekš sadefinētas darbības rezultāts ir tāds pats, vai netika pazaudētas kādas lietojumprogrammatūras iespējas.

5.2.7 Veiktspējas optimizēšana

Atšķirība no Oracle Adabas D nav iespēju sadalīt tabulu vairākās daļas, kā arī netiek izmantoti norādījumi SQL vaicājumos. Taču visticamāk ja notika tāda migrācija, tad datu bāzes apjoms ir diezgan neliels un tādēļ nevajadzētu būt sliktai veiktspējai. Ja tomēr kādas darbības notiek pārāk lēni, var pamēģināt optimizēt SQL vaicājumus vai uztaisīt tabulās papildus indeksus. Ja tabulā ir kolona ar primāro atslēgu un vaicājumā meklēšana notiek pēc šīs atslēgas, tad labāk ir pierakstīt atslēgas vārdu KEY. Piemēram:

```
SELECT DIRECT name,zip,address
INTO :hname,:hzip,:hstreet
FROM hotel
KEY  hno = :hno;
```

5.3 Secinājumi

Datu bāzes migrācija no Oracle uz Adabas D ir laba ar:

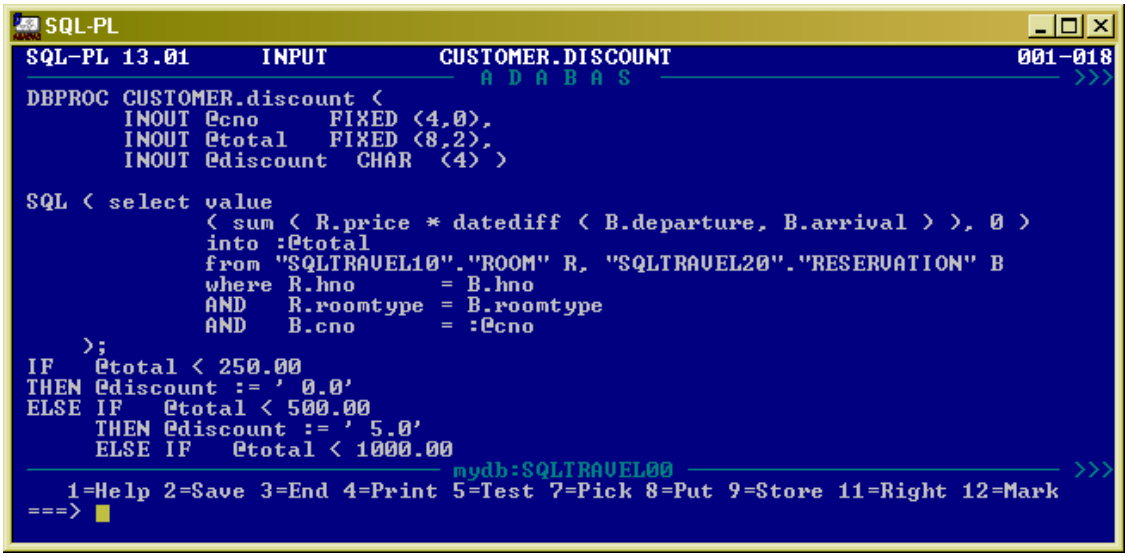
- Pati Adabas D DBPS ir lētāka par Oracle;
- Uzturēšanai Adabas D neprasa tik daudz resursu, cik to vajag Oracle datu bāzēm;
- Administrēšanā Adabas D arī ir vieglāka nekā Oracle;

Datu bāzes migrācija no Oracle uz Adabas D ir slikta ar:

- PL/SQL valodas konvertācija SQL-PL valodā ir apjomīgs un sarežģīts process, jo šīs valodas kaut arī ir līdzīgas, tomēr atšķirības tam ir pārāk lielas;

- Ir liela iespēja pazaudēt svarīgus datus, datu tipu neatbilstības dēļ, jo gandrīz visi Adabas D datu tipi atļauj mazākus pēc apjoma ierakstus;
- SQL-PL izstrādei pagaidām ir tikai viens rīks (Zīm. 5), ar ko principā nav ērti strādāt;
- Kamēr nav izlaista Adabas D 14.01 versija, tikmēr, ja sistēma bija UNICODE, ir liela iespēja pazaudēt vai sabojāt svarīgo informāciju;
- Pašā datu bāzē darbs ar XML failiem vairs nebūs iespējams;
- Trigerus vares izmantot tikai tabulām un skatījumiem, kas ir bāzeti uz vienas tabulas;
- Objektu tipus vairs nebūs iespējams uzturēt Adabas D.

Tātad var redzēt, ka migrācija no Oracle uz Adabas D, teiksim tā, nopietnai sistēmai ir apgrūtinoša un tā var neapmaksāties, jo zaudējot kādu jau strādājošu funkcionalitāti ir diezgan sāpīgi. Protams, ja sistēma neizmanto to funkcionalitāti, kas nebūs pēc Oracle datu bāzes, tad migrāciju var veikt un var būt tā pat apmaksāsies.



Zīm. 5 – Adabas D iebūvētais SQL-PL moduļu izstrādes rīks

6 Nobeigums

Bakalaura darbā izstrādes laikā tika izpētīts sekojošais:

- migrācijas process;
- ar to saistītas problēmas;
- Oracle PL/SQL valodas struktūra;
- Adabas D SQL-PL valodas struktūra;
-

Darbā izveidots detalizēts migrācijas procesa apraksts:

- no Adabas D uz Oracle, kur tika:
  - pārskaitīts kas ir jādara šādas migrācijas gadījumā;
  - izpētīta SQL-PL struktūru aizvietošana uz PL/SQL līdzīgām struktūrām, to līdzība un atšķirības;
  - piedāvāti šo problēmu apiešanas ceļi;
- no Oracle uz Adabas D, kur tika:
  - pārskaitīts kas ir jādara šādas migrācijas gadījumā;
  - izpētīta SQL-PL struktūru aizvietošana uz PL/SQL līdzīgām struktūrām, to līdzība un atšķirības;
  - piedāvāti šo problēmu apiešanas ceļi;

Secinājumi:

- Visslābāk būtu migrēt datu bāzi no Adabas D uz Oracle, jo neskatoties uz to, ka Oracle uzturēšana ir dārgāka, tomēr pašai DBMS ir daudz iespējas, kas nav Adabas D, piemēram, var strādāt ar lielākiem datu apjomiem nekā Adabas D (LONG -> CLOB), var izmantot UNICODE un tas ir ļoti svarīgi Latvijas projektiem, var izmantot tādu PL/SQL struktūru kā objektu tips, kas ir ērts JAVA lietošanai, var definēt trigerus uz datu bāzi vai shēmu, var strādāt ar XML dokumentiem un citas dažiem projektiem ļoti svarīgas lietas.
- Taču ja visas šis iespējas principiāli projektam nav vajadzīgas, tad var pāriet no dārgas Oracle DBPS uz vienkāršāku Adabas D DBPS. Principā

pēdējā Adabas D versija spēj pilnībā aizvietot Oracle 7 (vai daļēji vēlākas Oracle versijas).

- Tā, ka patreiz neeksistē tāds rīks, kas spētu automatizēt migrāciju uz Oracle no Adabas D un otrādāk, tad būs nepieciešams daudz laika un resursu migrācijas realizācijai. Neskatoties uz to, ka abām DBMS ir līdzīgas SQL programmēšanas valodas, tomēr to moduļi un struktūras atšķiras diezgan stipri un, piemēram, lai pārnest PL/SQL pakotni uz SQL-PL, būs jāizpēta pārēji PL/SQL datu bāzes moduļi, lai zinātu kādam ir jābūt beigu rezultātam SQL-PL valodā.

Kā nākamais darba pētītas problēmas risinājuma posms varētu būt migrācijas rīka izveide. Visticamāk rīks tiks veidots migrācijai no Adabas D uz Oracle DBPS. Šis bakalaura darbs varētu būt par pamatu šāda migrācijas rīka izveidei.

7 Literatūras saraksts

Grāmatu resursi:

1. *SQLWays™ White Paper: Database Migration Software* [tiešsaiste]. Ispirer Systems Ltd. [atsauce 20.04.2006]. Pieejams Internetā: http://www.ispirer.com/doc/sqlways_whitepaper.pdf

2. *Enterprise IT Migration for the Software AG Environment: Technical White Paper* [tiešsaiste]. BluePhoenix Solutions. 20.10.2005 [atsauce 22.04.2006]. Pieejams Internetā: http://www.bphx.com/docs/AdabasNatural_wp.pdf

3. *Database Migration: The Key to Unlocking your Business Assets* [tiešsaiste]. D. Fishman. DB Best Technologies. 2004 [atsauce 22.04.2006]. Pieejams Internetā: http://www.sql-server-performance.com/dbbest_Database_Migration_Unlocking_Business_Assets.pdf

4. *Oracle 9i Database Migration: Release 2 (9.2)* [tiešsaiste]. Oracle. 10.2002 [atsauce 23.04.2006]. Pieejams Internetā: http://www.lc.leidenuniv.nl/awcourse/oracle/server.920/a96530/toc.htm

5. *Database models* [tiešsaiste]. A. Lashenko. Toronto, Canada. UnixSpace. [atsauce 23.04.2006]. Pieejams Internetā: http://unixspace.com/context/databases.html

6. *Databse Migration solutions for Oracle* [tiešsaiste]. AdventNet, Inc. [atsauce 12.05.2006]. Pieejams Internetā: http://www.swissql.com/oracle-migration.html

7. *SQLWays* [tiešsaiste]. Ispirer Systems. [atsauce 12.05.2006]. Pieejams Internetā: http://www.ispirer.com/products/

11. *Oracle PL/SQL Language Pocket Reference* [tiešsaiste]. C. Dawes, S. Feuerstein, B. Pribyl. 05.1999 [atsauce 18.05.2006]. Pieejams Internetā: http://safari.oreilly.com/?XmlId=1-56592-457-6

12. *PL/SQL User's Guide and Reference* [tiešsaiste]. J. Russell. 12.2003 [atsauce 20.05.2006]. Pieejams Internetā: http://www.cs.umbc.edu/help/oracle8/server.815/a67842/toc.htm

13. *SQL Reference (Release 2)* [tiešsaiste]. D. Lorentz, J. Gregoire. 10.2002 [atsauce 20.05.2006]. Pieejams Internetā: http://www.lc.leidenuniv.nl/awcourse/oracle/server.920/a96540/toc.htm

14. *Adabas D SQL-PL* [tiešsaiste]. Software AG. 2001 gads [atsauce 25.05.2006]. Pieejams Internetā: http://portalclientes.consist.com.br/Portal_Arquivos/doc-eletronica/adabas/Adabas_D/html/sqlpleng4_1.htm

15. *Adabas D Reference* [tiešsaiste]. Software AG. 2001 gads [atsauce 25.05.2006]. Pieejams Internetā: http://portalclientes.consist.com.br/Portal_Arquivos/doc-eletronica/adabas/Adabas_D/html/refadeng4_1.htm

16. *Adabas D Load* [tiešsaiste]. Software AG. 2001 gads [atsauce 26.05.2006]. Pieejams Internetā: http://portalclientes.consist.com.br/Portal_Arquivos/doc-eletronica/adabas/Adabas_D/html/loadeng4_1.htm

Citi resursi:

8. *termina „migration" paskaidrojums* [tiešsaiste]. Lielā terminu vārdnīca DATORTERMINI. [atsauce 20.04.2006]. Pieejams Internetā: http://www.termini.lv/index.php

9. *SQL:1999, ранее известный как SQL3.* [tiešsaiste]. A. Eisenberg, J. Melton. ACM SIGMOD Record, Volume 28, Number 1. 03.1999 [atsauce 23.04.2006]. Pieejams Internetā: http://www.citforum.ru/database/digest/sql1999.shtml

10. *FAQ about Oracle Corporation (1.38)* [tiešsaiste]. F. Naude. 09.06.2005 [atsauce 4.05.2006]. Pieejams Internetā: http://www.orafaq.com/faqora.htm

8 Patstāvības apliecinājuma forma

Apliecinājums

Ar šo es apliecinu, ka šodien iesniegto bakalaura darbu es esmu veikusi pašrocīgi un esmu izmantojusi tikai tajā norādītos palīglīdzekļus.

Rīga, Paraksts:

9 Reģistrācija lapa

Bakalaura darbs izstrādāts
LU Datorikas nodaļā

Autors:
Fizikas un matemātikas ……
fakultātes studente Irina Gerasimenko
St. apl. Nr. Prog020007 2006. g. …. . jūnijā

Darba vadītājs:
Dr. sc. comp., LU lektors Juris Strods ……
LU Fizikas un matemātikas fakultāte

Recenzents:
Mg. sc. comp., LU lektors Aivars Niedrītis ……
LU Fizikas un matemātikas fakultāte

Darbs iesniegts Datorikas nodaļā 2006. g. … . jūnijā
Pieņēma sekretāre ……

Aizstāvēts datorzinātņu bakalaura pārbaudījumu komisijas sēdē
2006.g. ……. ar atzīmi ……
Protokols Nr.____

Bakalaura pārbaudījumu ……
komisijas sekretārs: ……