LATVIJAS UNIVERSITĀTE
FIZIKAS UN MATEMĀTIKAS
DATORIKAS NODAĻA

SQL optimizēšana meklēšanas formai

BAKALAURA DARBS

Autors: **Andris Tretjakovs**
Stud. apl. Datz030062
Darba vadītājs: lektors M.dat Aivars Niedrītis

RĪGA 2007

ANOTĀCIJA

Darba temats ir SQL vaicājumu veiktspējas uzlabošana. Šajā gadījumā vaicājums ir Latvenergo norēķinu sistēmas meklēšanas formas pieprasījums datu bāzei, kuram jāatgriež klienti vai to objekti, kas atbilst izvirzītajiem nosacījumiem. Sākot darbu pie šīs problēmas izvirzītie mērķi bija novērst kļūdas atgriežamajos datus un panākt, lai dati tiktu atgriezti tik ātri, lai rezultātus varētu izmantot darbinieki, kas strādā klientu apkalpošanas centrā. Kā darba rezultāts tika iegūta pakotne, kas meklētos rezultātus ieliek tabulā, kura tiek izmantota rezultātu attēlošanai. Pēc uzlabojumu uzstādīšanas, ja ievadītie parametri ir pietiekoši precīzi, rezultāti tiek atgriezti gandrīz uzreiz pēc meklēšanas uzsākšanas.

Atslēgvārdi: Oracle SQL optimizācija, saistītie mainīgie.

ABSTRACT

This document is about performace tuning of SQL statements. In this case statement comes from Latvenergo billing system as request for data about clients or objects who meets all search criteria. Goals of this work were to fix incorrect returned data and change request statement so result could be used in client serving center. As a result of this work is PL/SQL package which inserts all requested rows in temporary table so it can be displayed to employee who needed it. After patch was aplied, search form returned all rows in less than 1 second after search was launched ifcriteria were enaught good to filter data.

Keywords: Oracle SQL performance tuning, bind variables.

SATURA RĀDĪTĀJS

IEVADS

Šajā darbā apskatāmo formu paredzēts izmanot Latvenergo klientu apkalpošanas sistēmas ietvaros, kas pamatā ir balstīta uz Oracle E-Business Suite ar Oracle 9i datu bāzi. Lielākā daļa no sistēmas formām, tabulām un skatiem ir standartfunkcionalitāte, kas papildināta ar personalizācijām un pielāgota dažām Latvenergo darbības īpatnībām. Šobrīd tam tiek izmantotas divas citas formas: viena no tām ir paredzēta tehnisko pakalpojumu veidošanai, tādēļ tajā ir ļoti plašas iespējas meklēt pēc dažādiem tehniskiem parametriem, piemēram, elektroenerģijas skaitītāja tipa. Taču tajā nepastāv iespējas navigēt no atrastajiem rezultātiem tieši uz formām, kurās atrasos ierakstus var apskatīt sīkāk. Otra ir Oracle eBS universālā meklēšanas forma. No šīs formas atrastajiem rezultātiem ir iespējams pāriet uz formu, kurā notiek klienta datu uzturēšana. Tomēr šajā formā var meklēt tikai pēc klienta adreses un tieši klienta datiem – vārda, uzvārda, nosaukuma, personas koda un citiem, un tajā pastāv iespēja atvērt atrastajiem klientiem atsevišķu formu to datu uzturēšanai. Apskatāmajā formā pastāv navigācijas iespējas uz klienta datu uzturēšanu, līguma datu uzturēšanu un objekta tehnisko datu uzturēšanu. Kaut arī forma ir izstrādāta sen, tā nav ieviesta produkcijā, jo pastāvēja problēmas ar datu korektu atlasi un attēlošanu, kā arī, pēc pēdējo datu konvertācijas, problēmas radās ar formas veiktspēju. Tas neatbilst prasībām, jo viens no formas lietojumiem ir paredzēts klientu apkalpošanas centrā, kur tiek atbildēts uz saņemtajiem zvaniem. Darba mērķis ir panākt, lai meklējot pēc jebkura parametra formā rezultātus atgrieztu ātri un lai nepastāvētu tādas parametru kombinācijas, kas formas darbību paralizē. Formai ir paredzamas izmaiņas gan vizuālajā izskatā, gan datu meklēšanas algoritmā. Tā kā šis darbs ir par veiktspēju, tad tiek apskatītas tikai algoritma izmaiņas un risinājums ir izstrādāts tā, lai darbotos gan ar formas veco versiju, gan atjaunoto, kurā ir paredzēts palielināt parametru skaitu, pēc kuriem var veikt meklēšanu.

Darba izstrādes laikā galvenās vadlīnijas bija Oracle dokumentācija. Parasti veiktspējas problēmas ir unikālas, jo katram gadījumam ir citi apstākļi un dati, tadēļ papildus informācija tika meklēta konkrētos problēmām, vai risinājuma metodēm, kuras būtu pielietojamas. Darba pirmajā nodaļā apskatīta SQL vaicājuma drošība, jo pirms labojumu veikšanas formā aprakstīto uzbrukumu viedus bija iespējams veikt. Otrajā nodaļā ir aprakstītas dažas metodes, kas var dot ieguldījumu veiktspējas uzlabošanai, savukārt, trešajā nodaļā ir aprakstīta konkrēto risinājumu pielietošana un daļēji to rezultāti.

1. SQL DROŠĪBA

Uzbrukumi pēc sava veida ir vienkārši, jo uzbrucējs aplikācijai ievadē padod speciāli sagatavotu simbolu virkni ar vēlmi izpildīt sev vēlamo SQL vaicājumu. Šāda veida uzbrukumi ir vieglāk veicami pret atvērtā koda aplikācijām, jo uzbrucējs pirms tam var izanalizēt koda darbību un attiecīgi paredzēt, kādā veidā būtu jāsagatavo uzbrukuma kods. Tiem ir četras galvenās kategorijas:

1)SQL manipulācija

2)Koda ievietošana

3)Procedūras izsaukuma ievietošana

4)Atmiņas bufera pārpilde

Pirmie divi uzbrukumi veidi var tik vērsti ne tikai pret Oracle, bet arī citām datu bāžu sistēmām. SQL manipulācijas parasti tiek veiktas ar kopu operāciju palīdzību, piemēram, UNION vai arī mainot WHERE nosacījumus, lai vaicājums atgrieztu citu rezultātu. Pazīstamākais no tiem ir lietotāja identifikācijas vaicājuma nosacījuma papildināšana, lai tas vienmēr būtu patiess. Piemēram, vaicājumā

```
SELECT * FROM users
WHERE username = 'bob' AND PASSWORD='mypassword'
```

uzbrucējs “mypassword” vietā ievada “mypassword' or 'a'='a”, kā rezultātā vaicājums izskatīšies šādi:

```
SELECT * FROM users
WHERE username = 'bob' AND PASSWORD='mypassword' or 'a'='a'.
```

Izmantojot kopu operācijas, gala mērķis ir panākt, lai kopā ar paredzētajiem rezultātiem, tiktu atgriezti dati no citām tabulām. Ja aplikācija izpilda šādu vaicājumu:

```
SELECT product_name FROM all_products
```

WHERE product_name like '%CHAIRS%', uzbrucējs var, lietojot UNION, sakumu vaicājumu pārveidot uz šādu:

```
SELECT product_name FROM all_products
WHERE product_name like '%CHAIRS%' UNION SELECT username FROM dba_users
        WHERE username like '%'.
```

Šāds pieprasījums atgriezīs ne tikai paredzētos datus, bet arī visus lietotājvārdus.

Koda ievietošana mēģina pievienot papildus SQL pieprasījumus vai komandas eksistējošam vaicājumam. Šādi vaicājumi Oracle serveriem ir mazāk aktuāli kā Microsoft SQL serveriem, jo

Javā un PL/SQL Oracle neatbalsta vairākus SQL vaicājumus vienā pieprasījumā. Tomēr, ja tiek izpildīti ģenerēti PL/SQL bloki, tad tie var kļūt par šāda uzbrukuma mērķiem. Ja šādā veidā tiek izpildīts sekojošais kods:

BEGIN ENCRYPT_PASSWORD('bob', 'mypassword'); END;

tad uzbrucējs ar mēģināt "mypassword" vieta ievadīt virkni, lai izpildītu papildus komandu:

BEGIN ENCRYPT_PASSWORD('bob', 'mypassword');
DELETE FROM users WHERE upper(username)= upper ('admin'); END;

Bufera pārpildes iespējas ir atklātas vairāku datu bāžu funkcijās. Tai skaitā arī dažas Oracle funkcijas neatjauninātās datubāzes var būt jūtīgas pret bufera pārpildi, piemēram, tz_offset, to_timestamp_tz, bfilename. Lielākā daļa aplikāciju serveru neapstrādā gadījumus, kad tiek zaudēts savienojums ar datubāzes serveri dēļ šis pārpildes, un process darbojas līdz tiek pārtraukts savienojums ar klientu, kas šāda veida uzbrukumus dara efektīvus, kā pakalpojumu atteices uzbrukumus. Šīs bufera pārpildes var izraisīt izmantojot procedūru izsaukumu ievietošanu.

Funkciju vai procedūru ievietošana ir dažādu Oracle funkciju ievietošana ievainojamos SQL vaicājumos. Šāds uzbrukumu veids ir iespējams, Oracle datu bāze atļauj izpildīt funkcijas kā vaicājuma daļu, piemēram, iegūst kādu parametru. Funkcijas, kas tiek izpildītas kā vaicājuma daļa, nevar veikt izmaiņas datos, ja vien nav atzīmētas kā "PRAGMA TRANSACTION", bet tāda nav neviena no standarta Oracle funkcijām. Tomēr šīs funkcijas var izmantot, lai, piemēram, nosūtītu datus no datu bāzes uz attālinātu datoru. Sekojošo vaicājumu:

SELECT TRANSLATE('user_input', '1234567890qwertyuiop','123456789') FROM DUAL;

uz šādu:

SELECT TRANSLATE ('||UTL_HTTP.REQUEST('http://123.23.23.23/')||',
'1234567890qwertyuiop', '123456789') FROM DUAL.

Šis vaicājums pieprasīs lapu no Interneta servera. Uzbrucējs var manipulēt simbolu virkni un Interneta saiti tā, lai iekļautu citas funkcija, lai iegūtu datus un nosūtītu tos uz norādīti saiti, piemēram, PHP lapām saitē var norādīt parametrus, kas tiek nosūtīti uz serveri. Avotā [2] tiek apskatīti gadījumi, ka līdzīgā veidā kā iepriekš, ir iespējams modificēt dinamiski ģenerētus kursorus PL/SQL kodā. Tiem pamatā arī speciāli sagatavotas simbolu virknes padošana ievadā, kuru iekļaujot vaicājumā netiek interpretēta vairs kā virkne, bet gan kods.

No šīm problēmām palīdz tikt vaļā saistīto mainīgo izmantošana, jo Oracle datu bāze tos interpretē kā simbolu virknes, neatkarīgi no tā kādi simboli tajā ir. Protams, ir svarīgi tas, ka tie tiek lietoti pareizi, piemēram, ja lietotāja ievadītajam parametram ir nepieciešams abās pusēs ievadīt

“%” zīmes, tad nepareizais variants ir pievienot šos simbolus vaicājumā izskatās šādi:

```
        String name = request.getParameter("name");
conn.prepareStatement("SELECT id FROM users WHERE name LIKE '%" + name + "%'");
```

Tam labāk ir izmantot sekojošo sintaksi:

```
        String name = request.getParameter("name");
name = query.append("%").append(name).append("%");
pstmt = conn.prepareStatement("SELECT id FROM users WHERE name LIKE ?");
pstmt.setString (1, name);.
```

Šie piemēri ir ņemti no [1].

2.1 Pilna tabulas pārlase vai ieraksta atrašana ar indeksu

Grāmatas [3] autors iesaka biežāk lietot iekļautos ciklus un indeksus, jo parasti tiek prasīts neliels ierakstu skaits, pretējā gadījumā var zust atlasīšanas jēga, jo ierakstu skaits var būt tik liels, ka vairs nav lietojams. Bieži mēģinot uzlabot vaicājumu ātrdarbību ir jāizvēlas lietot indeksus vai arī veikt pilnu tabulas pārlasi. Indeksu lietošanai tiek uzskatītas vairākas priekšrocības.

Indeksu lasīšanas gandrīz vienmēr ir speciālajā atmiņā.

Lasīšana pēc indeksa pārbauda tikai nelielu daļu no katra bloka, tikai rindas, kuras ir nepieciešamas, nevis visas, tādā veidā taupot procesora laiku.

Lasīšanu pēc indeksa mazāk ietekmē tabulas izmēra pieaugums.

Atsevišķam blokam ir daudz lielāka iespējamība būt speciālā atmiņas apgabalā, kas paredzēts biežāk lietojamu datu glabāšanai, lai tie notiktu lasīti no cietā diska, salīdzinot ar vairāku bloku grupu.

Diska iekārtas parasti līdz ar atsevišķā ieraksta ielasīšanu ielasa arī blakus ierakstus atmiņā, tādējādi arī šie ieraksti vairs nebūs atsevišķi jālasa no diska.

Šīs grāmatas autors iesaka lietot indeksus, ja no tabulas nepieciešami mazāk kā 0.5% ierakstu, pilnu tabulas pārlasi, ja nepieciešamo ierakstu skaits pārsniedz 20%. citi gadījumi jāskatās pēc apstākļiem. Šī risinājuma ietvaros vienmēr tiek izvēlēta pieeja pēc indeksa, jo jebkuras tabulas pilna lasīšana izpildes laiku palielinātu, jo no iekļautajām tabulām pilna tabulas ielasīšana mazākajai prasa ap 30 sekundēm un lielākajai ap 5 minūtēm.

2.2 Saistīto mainīgo lietošana

Tā kā parasta vērtību iekļaušana vaicājumos, iekļauj iepriekšminētos drošības draudus un katrs vaicājums ir jāapstrādā kā jauns, jo iespēja, ka tas būs izpildīts tieši ar šādām vērtībām ir niecīga, tika apskatīta iespēja lietot saistītos mainīgos. Kā arī šādas metodes lietošana atļauj vienkāršāk meklēt specifiskus nosaukumus, piemēram, “Terry's backery”. Es gan nepārbaudīju vai datu bāzē glabājas kāds nosaukums ar šo vienu pēdiņu, taču pieļauju domu, ka tādi nosaukumi vai adreses, kas uzrakstītas, lietojot šo simbolu, varētu eksistēt. Pretējā gadījumā šāda veida nosaukumus speciāli sagatavot izmantojot TRANSLATE vai līdzīgu konstrukciju. Pirms labojumu veikšanas, ievadot šādu nosaukumu, forma izdeva ārā kļūdu par nekorektu SQL sintaksi, tādēļ tika meklēta papildus informācija par iegūstamajām priekšrocībām. Materiālā [4] tika dota tālāk

izklāstītā informācija.

Parasti SQL vaicājumu izpildē tiek veikti vairāki soļi:

1. pirmais solis jeb “hard parse”, kas pārbauda sintakses pareizību un tabulu, lauku eksistenci
2. otrais solis jeb “soft parse” sasaista vaicājumu ar konkrēto sesiju un veic pieejas tiesību pārbaudi
3. optimizēšana, kurā Oracle meklē labāko veidu, kā izpildīt vaicājumu, lietojot tā rīcībā esošos datus par izmantotajiem datu bāzes objektiem
4. vaicājuma izpilde, kas veic datu lasīšanu
5. vaicājuma datu ierakstu atgriezšana aplikācijai.

Pirmajos trīs soļu, kaut arī paši dati nav iesaistīti, izpilde prasa zināmu laiku, ko var ietaupīt izvairoties no šiem soļiem pēc iespējas vairāk. Galvenais Oracle mehānisms , kas ļaut to darīt, ir izpildāmo SQL vaicājumu bibliotēka. Pirmā soļa laikā tiek pārbaudīts vai šāds vaicājums jau ir bibliotēka, ja tas tajā atrodas, tad uzreiz var pāriet uz otro soli, izlaižot sintakses un kolonnu pārbaudi. Otrajā solī pārbauda pieejas tiesības un sesijas mainīgos un pēc tam izlaiž trešo soli, pārejot pie paša vaicājuma izpildes. Ja vaicājums netika atrasts bibliotēkā (tas tiek izpildīts pirmo reizi), tad jāveic visi iepriekšminētie soļi. Tā kā vaicājumu salīdzināšana notiek tāpat kā simbolu virknēm, tad par vienu un to pašu vaicājumu tiks uzskatīti tikai pilnībā identiski vaicājumi. Gadījumos, ja parametra vērtība tiek iekodēta pašā vaicājumā, tad attiecīgi divi līdzīgi vaicājumi, kuriem atšķirsies tikai šī iekodētā vērtība, tiks uzskatīti kā divi dažādi un tiks veikti visi iepriekšminētie soļi. Izmantojot saistītos mainīgos vaicājumi būs vienādi un visi soļi tiks veikti tikai pirmajā reizē. Mainīgo vērtības vaicājumā ievieto tā izpildes brīdī.

Kaut arī gadījumos, kad vaicājumā tiek precīzi iekodētas vērtības, tie tiek uzskatīti par atšķirīgiem(Oracle dokumentācija). Izņēmuma situācija ir tad, ja parametram CURSOR_SHARING ir uzstādīta vērtība SIMILAR vai FORCE (noklusētā vērtība EXACT). Šajos gadījumos tiek pārbaudīts vai vaicājumam ir jau identisks bibliotēkā, ja nav tad meklēti līdzīgie Vaicājumi tiek uzskatīti par līdzīgiem, tad ja tie ir identiski izņemot dažu parametru vērtības. Ja mainīgā vērtība ir EXACT, tad tiek pārbaudīti tikai identiskie vaicājumi, pretējā gadījumā tiek pārbaudīti arī līdzīgie. Starpība starp SIMILAR un FORCE ir tādā, ka pirmais darbojas bez izpildes plāna pārveides. Otra vērtība var mainīt plānus, tādēļ to ir ieteicams lietot tikai tad, kad risks, ka plāns nav optimāls, ir lielāks par ieguvumiem no šo vaicājumu atkārtotas izmantošanas. Tomēr priekšroka ir dodama šāda atkārtojami izmantojam koda rakstīšanai uzreiz, nevis paļauties uz CURSOR_SHARING parametru, jo tas uzliek papildu slodzi pārbaudot, vai tāds vaicājums nav saglabāts bibliotēkā. Šis parametrs būtu lietojams jau izstrādātai aplikācijai, lai kaut daļēji iegūtu uzlabojumos, kas tiek

panākti izmantojot saistītos mainīgos.

2.3 Histogrammas

Izpildes plānu precizitāte ir atkarīga no tā cik precīzi, ģenerējot izpildes plānu, izdodas paredzēt izmantoto nosacījumu selektivitāti. Parasti par tabulu savāktie dati satur ierakstu skaitu, dažādo vērtību skaitu, maksimālo un minimālo vērtību un ierakstu skaitu ar NULL šajā kolonnā. Precīzāku tabulas datu sadalījumu var iegūt no histogrammām [5]. Histogrammas tiek lietotas tikai bieži WHERE nosacījumos lietojamām kolonnām tad, ja datu sadalījums ir nevienmērīgs. Gadījumos, ja dati pēc vērtībām ir sadalījušies vienmērīgi, optimizators spēj diezgan precīzi noteikt atgriežamo ierakstu skaitu arī bez histogrammu lietošanas. Tā kā tas ir statiskas, tad, ja tabulā notiek lielas datu izmaiņas, tās ir jāatjauno. Histogrammas nav noderīgas, ja:

visi predikāti šai kolonnai lieto saistītos mainīgos(šī forma bija vienīgā, ko man bija izdevies redzēt, kura neizmanto saistītos mainīgos)

kolonnas dati ir sadalīti vienmērīgi

kolonna ir unikāla un to lieto tikai ar vienādības predikātiem.

Histogrammas tiek veidotas izmantojot pakotni DBMS_STATS:

EXECUTE DBMS_STATS_GATHER_TABLE_STATS {'scott','tabulas_nosaukums', METHOD_OTP=>'FOR COLUMNE SIZE 10 kolonnas_nosauk'}; Size 10 norāda, cik daļās(paciņās) tiks sadalīts viss intervāls. Parasti tas tiek norādīts 75, taču ar šo parametru ir jāeksperimentē katrā gadījumā atsevišķi, lai atrastu labāko izpildes plānu. Histogrammām ir divi tipi, vienas ir balstītas uz vērtību(biežumu), otra uz skaitu. Pirmā ir lietojama tad, ja dažādo vērību skaits kolonnā ir mazāks par paciņu skaitu. Šādā gadījumā visām vērtībām ir atbilstoša paciņa, kas satur datus par to, cik bieži šī vērtība atkārtojas. otrs histogrammu tips katrā paciņā satur aptuveni vienādu vērtību skaitu, katra paciņa satur lielāko vērtību, tādējādi tiek iegūts aptuvens priekšstats par vērību sadalījumu. Piemēram, ja ierakstu skaits pēc vērtībām ir šāds: 1589464 ierakstu ar vērtību 10, 13091 - ar vērību 18 un 21889 ierakstu ar NULL. Ar histogrammu meklējot pēc vērtības 18, aprēķinot selektivitāti iegūst S=13091/(13091+1589464)= 0,008. Ja, netiek izmantota histogramma, tad tiek uzskatīts, ka selektivitāte ir 50 procenti. Ja šī tabula būtu indeksēta, tad pirmajā gadījumā tas tiktu izmantots, otrajā gadījumā tiktu lasīti visi tabulas dati, kas ņemot vērā patiesu ierakstu skaitu ar vērtību 18 nav tas labākais un ātrākais.

2.4 Vaicājumu norādes

Oracle piedāvā iespēju modificēt izpildes plānu izmantojot norādes, to skaits ir liels(arī

nedokumentēto), taču darbā es apskatu tikai dažas, kas iekļautas Oracle dokumentācijā[5]. Tās parasti tiek izmantotas gadījumos, ja vaicājuma autoram ir zināms kas vairāk par to tabulu datiem nekā Oracle optimizatoram, piemēram, iepriekšminētajā piemērā ar histogrammām, parastā gadījumā balstoties uz statistikām tiks veikta pilna tabulas datu nolasīšana, jo tiek uzskatīts, ka meklētajam parametram atbilst puse tabulas ierakstu, taču zinot, ka patiesībā šādam nosacījuma atbilst daudz mazāk ierakstu, mēs varam pateikt, lai tiktu izmantots indekss. Tomēr norāžu lietošana jāveic ļoti izmanīgi, jo iedodot norādi ar pareizu sintaksi, pārējais plāns tiks veidots balstoties uz to arī tad, ja tā ir kļūdaina pēc loģikas, piemēram, norādīts nepareizais indekss. Tālāk tiks apskatītas dažas no šīm norādēm sīkāk.

ALL_ROWS un FIRST_ROWS(n), kur n vesels skaitlis lielāks par 0. pirmais liek Oracle optimizēt vaicājuma izpildes plānu tā, lai pēc iespējas ātrāk tiktu atgrieztas visi ieraksti, bet otrs, lai pēc iespējas ātrāk atgrieztu pirmos n ierakstus. FIRST_ROWS tiek ignorēts, ja vaicājumā ir operatori, kuriem nepieciešama visu ierakstu nolasīšana, piemēram, ORDER BY, DISTINCT, GROUP BY, FOR UPDATE, agregātfunkcijas kā SUM un kopu operatori. Nākošās norādes regulē pieejas veidu tabulām. Biežāk lietotie no šīs grupas ir FULL un INDEX. No tabulas vai tās aliasa, kas norādīts iekavās aiz FULL, ieraksti tiks nolasīti visi pilnībā. Zināmā mērā pretēja norādē ir INDEX, kur iekavās norāda tabulu vai tās aliasu un indeksu, kuru jāizmanto. Šajā gadījumā ieraksti tiks meklēti izmantojot norādīto indeksu. Šo mēdz lietot, lai, piemēram, vairāku indeksu gadījumā, tiktu izmantos tas, kas konkrētajā situācijā ir labāks un nodrošina lielāku selektivitāti. INDEX_FFS norāda, ka pilna lasīšana jāveic indeksam. To var izmantot gadījumos, ja visi nepieciešamie dati atrodas indeksā un nav nepieciešams griezties pie tabulas. Ieguvums no šīs norādes ir gadījumā, ja atgriežamo ierakstu skaits ir liels. Šo norādi es izmantoju risinājumā, lai ātrāk atrastu pēc adreses gadījumā, ja meklējamā virkne sākas ar "%" vai arī tā ir ļoti tuvu sākumam. Ar NO_INDEX var norādīt, kādu indeksu neizmantot, gadījumos, ja tabulai ir vairāki indeksi un tos var izmantot datu atlasei, tad tiks izmantots kāds no atlikušajiem.

Kā piemērus no norādēm, kas ietekmē vaicājumu pārveidi var minēt MERGE un NO_MERGE. Norādot NO_MERGE un aiz tā iekavās skatu vai tā aliasu(var būt arī iekšējs vaicājums), tas netiks sapludināts ar pārējo vaicājumu. Šo norādi esmu lietojis gadījumos, kad iekšēja vaicājumā esmu ielasījis ierakstus, kurus man vajag un tad pie tā pievienot aprakstošos datus no citām tabulām, piemēram, līguma klientu. Gadījumā, ja netika izmantota šī norāde, šie papildus dati tika meklēti vēl tad, kad nebija pilnībā atfiltrēti liekie ieraksti. MERGE var izmantot, lai skatu, kas izmantots vaicājumā organiski iekļautu vaicājumā, veicot nepieciešamo ierakstu atlasi ātrāk.

Tālāk sekojošās norādes ietekmē tabulu savienošanu. Kā pirmo var minēt ORDERED. Šajā

gadījumā tabulas tiks savienotas tādā secībā, kādā ir norādītas. Ja kāda iemesla dēļ tabulas nav norādātas pareizā secībā, ir iespējams iegūt CARTESIAN JOIN, kas nozīmē, ka tabulas katrs ieraksts tiks savienots ar otras tabulas katru ierakstu. Rezultātu pareizību tas neietekmēs, taču, ja tabulas ir lielas, ļoti ticams, ka vaicājums datus atgriezīs pēc ļoti ilga laika. Piemēram, ja jau ir atlasīti 100 ieraksti un tiem kļūdaini pievienot tabulu ar 10000 ierakstiem, tad līdz lieko datu atfiltrēšanai būs 1000000 ierakstu. Tā kā parasti tabulas ir lielākas, tad iegūtais rezultāts var nesaiet operatīvajā atmiņā un tad tiek glabāts uz cietā diska, kas izpildes laiku palielina vēl vairāk. Ja nav nepieciešams norādīt visu tabulu secību, bet tikai tabulu ar kuru sākt var izmantot LEADING un iekavās tabulas nosaukumu.

USE_NL norāda veidu, kā tabulām tikt savienotām. Tabulas var savienot vairākos veidos un USE_NL pasaka, ka jāizmanto iekļautie cikli. Šis ir parasts veids, kā tabulas tiek savienots, ja atgriežamo ierakstu skaits ir mazs vai arī nepieciešams pēc iespējas ātrāk atgriezt pirmos rezultātus, jo šajā gadījumā katram pirmās tabulas ierakstam tik līdz tas tiek iegūts, tiek piemeklēts atbilstošs ieraksts no otrās tabulas. USE_HASH gadījumā abas tabulas sākuma nolasa pilnībā(vai arī indeksus pēc kuriem tiek savienotas tabulas) un tad visi ieraksti tiek savienoti uzreiz. Rezultātā visi savienotie ieraksti tiks iegūtu praktiski vienlaicīgi. Starpību starp USE_HASH un USE_NL uzskatāmi var redzēt atlasot, piemēram, SQL Navigatorā vairāk kā 250 ierakstus, neizmantojot Fetch all funkciju. USE_NL gadījumā pirmie 250 ieraksti tiks atgriezti ātrāk kā ar USE_HASH, taču kad mēģinās paskatīt nākošos 250 ierakstus, nāksies gaidīt aptuveni tik pat ilgi, ka pirmos 250 ierakstus(uz servera vaicājums ir apturēts līdz tālākajām lietotāja darbībām), taču ar USE_HASH nākošie ieraksti būs redzami uzreiz.

No citām norādēm populārākais varētu būt APPEND. Tas tiek izmantots INSERT operācijās, lai datus tabulā ieliktu nevis parastajā veidā, bet uzreiz rakstītu datu failos, neko neievietojot operatīvajā atmiņā, līdzīgi kā SQL*Loader. Tas paātrina datu ievietošanu, taču, lai to varētu veikt, jāizpildās nosacījumiem: tabula nevar būt daļa no klastera vai būt kārtota pēc indeksa, tabula nevar saturēt objektu tipu kolonnas, tabulai, kurā dati tiek ievietoti, nevar būt trigeri vai konstrainti, transakcija ar šāda veida datu ievietošanu nevar tikt sadalīta, nevar tikt izpildīti vaicājumi, kas izmanto datus šajā paša transakcijā, kurā dati ir ievietoti[6]. Ttiešā ievietošana atbalsta tikai apakšvaicājuma sintaksi, nevis VALUES[7]. Kā vēl vienu priekšrocību šādam ievietošanas veidam var minēt, ka ir iespējams atslēgt ierakstu veidošanu žurnālā.

2.5 Indeksu organizētas tabulas

Viena no ER modelī iekļautajām tabulām(kns_ctr_counter_usage_items) ir nevis vienkārša

tabula, bet speciāls tabulas vieds, kas apskatīts avotā [1]. Indeksu organizētas tabulas ir nevis tabulas, bet indeksi, kas nenorāda uz tabulu. Gadījumos, ja indekss satur visas kolonnas, kas nepieciešamas vaicājuma izpildei, tad vēršanās pie atbilstošās tabulas nenotiek. Ja ir izveidots indekss, kas sevī iekļauj visas tabulas kolonnas, tad varētu atbrīvoties no pašas tabulas pilnībā. Šis tabulu veids to arī dara, ietaupot vietu un laiku. Tā kā šādam izveidotam indeksam nav tabulas ieraksta, uz kuru tas norādītu, tad nav nepieciešami arī ROWID, tādējādi ieliekot vairāk ierakstus katrā blokā kā parasts indekss. Šos indeksus ir vieglāk arī glabāt kešā, jo tie aizņem mazāk vietas. Pirms indeksu organizētas tabulas izmantošanas jāņem vērā vairāki nosacījumi:

rindas ir gandrīz tādā pašā garumā, ka to indeksa atslēga. Tabulās tie paši dati tiek glabāti efektīvāk un kompaktāk kā indeksā. Ja rindas ir garas, salīdzinoši ar tās indeksu, tad neliels indekss un tabulas ieraksta lasīšana pēc tā , strādās tik pat labi vai pat labāk un elastīgāk.

Ieraksts gandrīz vienmēr tiek sasniegts caur vienu indeksu, iespējams visu vai daļu no primārā indeksa. Indeksu organizētai tabulai var izveidot arī sekundāros un parastos indeksus, taču šādā gadījumā tiek iznīcināts iemesls šādas tabulas veidošanai.

Dažreiz ieraksti tiek lasīti lielos intervālos balstoties uz indeksa sakārtoto secību. Lielu intervālu lasīšana indeksā ir diezgan efektīva salīdzinot ar to pašu rindiņu ielasīšanu no parastas tabulas, jo tur tās ir izkaisītas. Šis faktors ir priekšrocība indeksu organizētajai tabulai, kaut arī ir pretrunā ar iepriekšminēto punktu.

Tabulas datu modificēšana- jaunu ierakstu veidošana, dzēšana, maiņa. Parastās tabulas šis darbības veic daudz labāk, tādēļ transakciju aplikācijās parasti neizmanto indeksu organizētās tabulas, jo dati ir mainīgi.

2.6 Izmaksu balstīta optimizatora darbība

Atšķirībā no uz noteikumiem balstīta optimizatora, uz izmaksām balstītajam optimizatoram nav viennozīmīga un ātra ceļa kā iegūt rezultātu [8]. Tas pielāgojas videi, bet tas ir iespējams tikai tad, ja statistikas par izmantojamajiem objektiem ir atjaunotas. Šī optimizatora darbību var nosacīti sadalīt vairākās daļās:

1. SQL parsēšana – pārbauda sintaksi, tiesības uz objektu
2. uzģenerē sarakstu ar visiem iespējamajiem izpildes plāniem
3. izrēķina aptuvenās izmaksas katram izpildes plānam izmantojot visu pieejamās objektu statistikas
4. izvēlas plānu ar vismazākajām izmaksām

Izmaksu optimizators tiks lietots tikai tad, ja vismaz vienai no izmantotajām tabulām ir

izveidotas statistikas vai arī tas tiks noteikts ar norādēm. Lai veiksmīgi to izmantotu, ir jāsaprot kā šis optimizators darbojas.

Primārā atslēga vai unikāls indeks ar vienādību – šādu indeksu selektivitāte tiek uzskatīta par 100%. neviena cita pieejas metode nedod tik precīzus rezultātus, tādēļ tas tiek izmantots, kad vien ir pieejams.

Neunikāls indekss ar vienādību – šādiem indeksiem tiek rēķināta selektivitāte. Optimizators pieņem, ka tabulā dati ir izvietoti vienmērīgi.

Intervāla nolasīšana – šajā gadījumā selektivitāte indeksam tiek noteikta izmantojot pēdējo lielāko un mazāko vērtību. Tāpat kā iepriekš, tiek pieņemts, ka dati tabulā izvietoti vienmērīgi.

Intervāla nolasīšana ar saistītajiem mainīgajiem – tāpat kā iepriekšējā gadījumā selektivitāte tiek minēta. Pirms Oracle9i saistīto mainīgo vērtības nebija redzamas parsēšanas laikā(tās tiek padotas pēc izpildes plāna izveides), tādēļ tiek pieņemts, ka selektivitāte ir 25%, ja tiek salīdzināts ar precīzu vērtību (WHERE dept_no=:b1) un 50% priekš intervāliem (WHERE dept_no>:b1 AND dept_no<:b2). Sākot ar Oracle9i versiju, optimizators iegūst saistīto mainīgo vērtības pirms izpildes plāna veidošanas.

Histogrammas – pirms Oracle7i, šis optimizators nespēja atšķirt dažāda biežuma sastopamus datus.

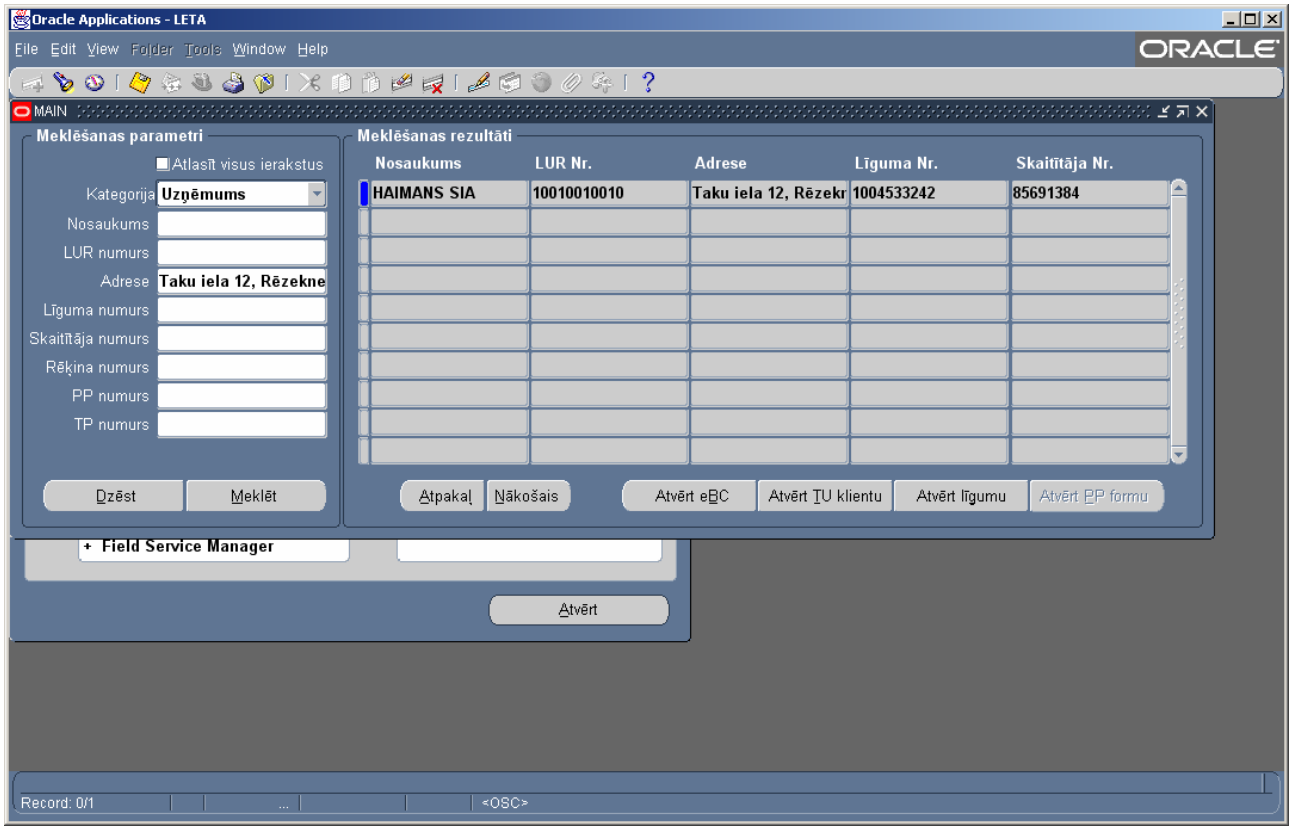
Sistēmas resursu noslodze – pēc noklusējuma uz izmaksām balstītais optimizators pieņem, ka datu bāzi izmanto tikai viens lietotājs. Oracle9i piedāvā iespēju saglabāt datus par sistēmas resursu noslodzi un tādā veidā izdarīt labāk izvēli ņemot vērā noslogojumu. To iespējams ir izdarīt izmantojot DBMS_STATS.GATHER_SYSTEM_STATS pakotni.

Esošo statistiku nozīme – izpildes plānam ir mazas izvēles iespējas, ja tabula ir izanalizēta, bet indekss nav, vai arī otrādi. Tādēļ nav ieteicams piespiest izmantot uz izmaksām balstīto optimizatoru, ja par nevienu no tabulām, kas iesaistītas vaicājumā, nav statistikas. Vecu statistiku lietošana var būt bīstamāka par to izrēķināšanu darbības laikā, tai paša laikā ir jāatceras, ka statistiku savākšana ir laikietilpīgs process, kas var būt kritiski daudzlietotāju sistēmām

3. RISINĀJUMI

3.1 Esošā situācija

Izmantojamā norēķinu sistēma darbojas uz Oracle9i datubāzes. Tas pamatā ir Oracle e-Business Suite, kas ārējās norēķinu sistēmas gadījuma ir papildināta ar trūkstošajām funkcionalitātēm, bet iekšējās tikai mazliet papildināta formu darbība. Šīs abas sistēmas izmanto vienu datu bāzi un brīžiem pārklājas. Tā kā pamatā ir Oracle izstrādājums gan tabulas , gan formas, tad, lai netiktu traucēta standarta funkciju darbība, ir jāiekļaujas tajā piedāvātājās iespējās. No tā seko, ka bieži par ārējām atslēgām kalpo papildinošie atribūtu lauki, ko speciāli paredzēti sākotnējās sistēmas pielāgošanai. To datu tips parasti ir varchar2, tādēļ, neuzmanīgi rakstot vaicājumu, var viegli palaist garām nepieciešamu datu tipa(parasti attiecas uz tabulu identifikatoriem) vai formāta(parasti datumiem) pārveidi. Kā piemēru var minēt tabulu hz_parties, kas satur ierakstus gan par klientiem (privātpersonas un juridiskie), gan objektiem(dzīvokli, mājas un citi). Tikai viena no šajā meklēšanā izmantotajām tabulām nav Oracle veidots datu bāzes objekts (KNS_CTR_COUNTER_USAGE_ITEMS). Parasti šādas tabulas ir veidotas jaunai funkcionalitātei, kas nav iekļauta Oracle e-Busness Suite. Pašas norēķinu sistēmas mērķis ir organizēt elektrības rēķinu izrakstīšana, jaunu pieslēgumu veidošana. Formā ir iespējams meklēt objektus, juridiskos klientus un privātpersonas. Izvēloties kādu no šiem tipiem, meklēšana sadalās atsevišķos zaros, kurā katrā ir savi meklēšanas parametri. Juridiskai personai tie ir: nosaukums, reģistrācijas numurs, adrese, līguma numurs, kāds no rēķina numuriem, transformatora punkta numurs, skaitītāja numurs, šim klientam kāda veidotā pakalpojuma pieprasījuma numurs. Privātpersonām meklēšanas parametri ir līdzīgi: vārds, uzvārds, personas kods, adrese, līguma numurs, kāds no rēķina numuriem, transformatora punkta numurs, skaitītāja numurs, šim klientam kāda veidotā pakalpojuma pieprasījuma numurs. Objektiem šo parametru ir mazliet mazāk: nosaukums, adrese, līguma numurs, transformatora punkta numurs, skaitītāja numurs, šim objektam kāda veidotā pakalpojuma pieprasījuma numurs. Tika izteikta vēlme pēc jauniem meklēšanas kritērijiem ar klienta vai objekta numuru sistēmā un sistēmas konta numuru klientiem. Viens no formas skatījumu ir pievienots mazliet zemāk. Nomainot kategoriju, mainās ievadāmo parametru lauku. Zem rezultātiem ir pogas navigācijai uz citām formā un poga “Nākošais” ar kuru tiek aiziets uzskatījumu, kur redzami visi šim klientam veidotie pakalpojuma pieprasījumi.



3.1. att. **Meklēšanas formas paraugs**

Pirms labojumu veikšanas meklēšana notika tālāk izklāstītajā veidā. Lietotājs ievada parametrus, pēc kuriem vēlas veikt meklēšanu un ko tieši vēlas meklēt. Forma šos parametrus padod tālāk pakotnei, kas vadoties pēc šiem parametriem, atlasa datus no iepriekš definēta skata un ieliekt īslaicīgā tabulā. Tālāk forma attēlo šīs tabulas datus lietotājam. Katra veida meklēšanai(privātpersona, juridiska persona vai objekts) ir savs skats, kas sevī iekļauj visas kolonnas, pēc kurām var veikt datu atlasi. Vēlamais mērķis bija pārstrādāt šo datu atlasi, jo, ņemot vērā datu apjomus un informāciju, kas tiek attēlota lietotājam kā rezultāts, šādu skatu lietošana formas darbību paralizē, piemēram, privātpersonas meklēšana pēc precīza skatītāja numura prasa 20 minūtes un ilgāk. Veiktspējas problēmu galvenie iemesli ir izmantotās tabulas un to savienošanas nosacījumi. Šajos pirms tam definētajos skatos bija iekļautas gandrīz visas tabulas, kas ir attēlotas pielikumā pievienotajā ER modelī, vienīgi iekļautas nebija okc_k_party_roles_b, ja notika objektu meklēšana, vai arī okc_k_items un okc_k_lines_b, ja meklēja klientus. Manā risinājumā tabulas, kas ir ārpus sarkanās līnijas, tiek pievienotas tikai tad, ja tiek meklēts pēc atbilstošajiem parametriem. Tā kā mazāka tabula ir okc_k_headers_b ar 1 miljonu ierakstu, lielākā csi_iea_values ar 17 miljoniem ierakstu un visas pārējas, izņemot ra_customer_trx_all ar 7 miljoniem ierakstu, ir robežās no 2 līdz 3 miljoni, tad kopējais apstrādājamo ierakstu skaits ir liels.

Tabula, kurā tiek glabāti atrastie rezultāti izskatās šādi:

```
NAME                           Null?     Type
------------------------------ --------- -----
CHR_ID                                   NUMBER
PARTY_ID                                 NUMBER
COUNTER_INSTANCE_ID                      NUMBER
CUSTOMER_TRX_ID                          NUMBER
PP_TASK_ID                               NUMBER
TP_ATTRIBUTE_VALUE_ID                    NUMBER
PERSON_FIRST_NAME                        VARCHAR2(100)
PERSON_LAST_NAME                         VARCHAR2(100)
PARTY_NAME                               VARCHAR2(100)
JGZZ_FISCAL_CODE                         VARCHAR2(60)
STATE                                    VARCHAR2(60)
PROVINCE                                 VARCHAR2(60)
FLOOR                                    VARCHAR2(60)
COUNTY                                   VARCHAR2(60)
SUITE                                    VARCHAR2(60)
STREET                                   VARCHAR2(60)
STREET_NUMBER                            VARCHAR2(60)
POSTAL_CODE                              VARCHAR2(60)
ADDRESS4                                 VARCHAR2(240)
ADDRESS                                  VARCHAR2(240)
CONTRACT_NUMBER                          VARCHAR2(100)
COUNTER_SERIAL_NUMBER                    VARCHAR2(100)
TRX_NUMBER                               VARCHAR2(100)
PP_NUMBER                                VARCHAR2(100)
TP_NUMBER                                VARCHAR2(100).
```

Lietotājiem tiek attēloti dati tikai no šādām kolonnām:person_first_name, person_last_name, party_name (uzņēmuma vai objekta nosaukums), jgzz_fiscal_code (personas kods privārpersonām un Latvijas Uznēmumu reģistra numurs juridiskajiem klientiem), contract_number un adreses lauki state, province, floor, county, suite, street, street_number, postal_code, address4 un address. Formā tika attēlots arī skaitītāja seriālais numurs, taču tas vairs netiek attēlots, lai meklēšanas rezultāti būtu pārskatāmi, jo dažiem lielajiem klientiem ir objekti ar daudziem skaitītajiem, piemēram, dažādas ražotnes, kas attiecīgi noveda pie tā, ka šis klients tika attēlots tik reizes, cik viņam skaitītāju.

3.2 Meklēšana pēc klienta datiem

Privātpersonu un juridisko klientu meklēšana notika gandrīz identiski, jo visas tabulas ir tās pašas, atšķiras vienīgi viens "where" nosacījums tabulai hz_parties, kas norāda klienta tipu, un, atlasot uzņēmumus, laicīgajā tabulā vārda un uzvārda vietā tika ielikta vērība NULL. Procedūra, kurai tika padoti parametri, pēc kuriem tiek meklēts, vaicājumu glabāja kā simbolu virkni, kuras beigās tika pievienoti ierobežojoši nosacījumi. Tad ar komandu "execute immediate" vaicājums tika izpildīts. Es gribēju realizēt meklēšanu tā, lai tiktu iesaistīts minimāls daudzums tabulu, atlasot tikai to, kas nepieciešams, t.i. klienta datus, līguma numuru un adresi(adrese ir arī tabulā hz_parties, taču tādā formātā pēc tās nav iespējams kārtot). Šīs tabulas ER modelī(1. pielikums) ir apvilktas ar

sarkanu krāsu, izņemot okc_k_party_roles_b, ja meklē objektus, vai arī okc_k_items un okc_k_lines_b, ja meklē klientus, jo kaut arī klienti un objekti glabājas vienā tabulā, atkarībā no to tipa, saistība ar līgumu tabula ir cita. Tāpat, lai būtu iespējams dinamiski papildināt, es izvēlējos to sadalīt loģiskās daļās: sākums, kas satur INSERT, daļa, kurā tiek glabāti norādes vaicājumi izpildei (HINT), kolonnas, kas jāatlasa, tabulas, kas izmantojamas, un divas ierobežojošo nosacījumu, viena no tām nemainās un glabā saites starp obligāti izmantojamām tabulām, bet otra tiks papildināta ar papildus nosacījumiem. Norāžu daļa jau tika plānota pašā sakumā, jo dažos gadījumos pēc pieredzes ar jau sākumā minēto universālo meklēšanas formu vaicājumu izpildes plāns neatbilda tam, kā dati atlasītos ātrāk. FROM daļā sākotnēji es izmantoju 3 tabulas – klientu dati, līgumi un adreses, kā arī divas starptabulas, kas glabā saistību starp šīm tabulām. WHERE nosacījumi izskatās sekojoši:

```
AND okh.sts_code IN ("ACTIVE", "QA_HOLD")
AND okp.chr_id = okh.id
AND okp.jtot_object1_code = "OKX_PARTY"
AND okp.rle_code = "CUSTOMER"
AND hp.party_id = to_number(okp.object1_id1)
AND hp.attribute_category = "BUSINESS"
AND pst.party_id = hp.party_id
AND pst.identifying_address_flag = "Y"
AND loc.location_id = pst.location_id
AND to_char(hp.party_id)=okp.object1_id1.
```

Nosacījums “hp.attribute_category = "BUSINESS" ” norāda klienta tipu, ja tiek meklēta privātpersona, tad BUSINESS vietā ir PERSON. Šajā daļā saite starp personu tabulu (hp) un personu-līgumu starp tabulu iekļauta divreiz, jo personu identifikators personu pēc datu tipa ir skaitlis un šādā veidā arī indeksēts, bet starp tabulā tas tiek glabāts ar varchar2 tipu un atbilstoši indeksēts. Gadījumos, ja iekļauj tikai vienu no šīm saitēm, tad līgumu un personu savienošanai kādā no virzieniem(no personas uz līgumu vai no līguma uz personu) ir nepieciešama tabulas pilna pārlase. Ja ir tikai pirmais no šiem nosacījumiem, tad pilnā pārlase veicama okc_k_party_roles_b (tabulā ir 2'141'394 ieraksti un pilnās pārlases izmaksas ir 4810) gadījumos, ja zināmai personai jāpiemeklē līgums. Pretējā gadījumā pilnā pārlase veicama tabula hz_parties, kurā ir 3'293'706 ierakstu un izmaksas šādai darbībai ir 21'906. Izmantojot šādus nosacījumus, jau ir iespējams meklēt pēc klienta datiem un līguma. Papildus nosacījumi tiek pievienoti izmantojot BIND mainīgos drošības(apskatīti pirmajā nodaļā) un veiktspējas apsvērumu dēļ, jo šāda veida SQL vaicājumu ir iespējams izmantot atkārtoti visām meklēšanām pēc līguma numura(šādā gadījumā

nav jāveic vaicājuma atkārtota sintakses pārbaude un izpildes plāna ģenerēšana). Šādā pašā veidā parametri tiek pievienoti, ja meklēšana notiek pēc uzņēmuma nosaukuma, uzņēmuma reģistrācija numura, sistēmas klienta numura, privātpersonas vārda un uzvārda, personas koda. Meklēšanā pēc klienta datiem izpildes plāns paliek tāds pats tikai mainās indekss, kas tiek izmantots, jo visas kolonna, kas satur šīs vērtības, ir indeksētas, kā neunikālas vērības, izņemot klienta numuru sistēmā. Personas vārds un uzvārds ir indeksēts katrs atsevišķi un kopā. Izpildes plāni, pēc kuriem tiek pildīti vaicājumi pievienoti zemāk.

SELECT STATEMENT ()		16
NESTED LOOPS ()		16
NESTED LOOPS ()		14
NESTED LOOPS ()		11
NESTED LOOPS ()		10
TABLE ACCESS (BY INDEX ROWID)	OKC_K_HEADERS_B	4
INDEX (RANGE SCAN)	OKC_K_HEADERS_B_U2	3
TABLE ACCESS (BY INDEX ROWID)	OKC_K_PARTY_ROLES_B	6
INDEX (RANGE SCAN)	KNS_OKC_K_PARTY_ROLES_B_N6	2
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTIES	1
INDEX (UNIQUE SCAN)	HZ_PARTIES_U3	
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTY_SITES	3
INDEX (RANGE SCAN)	HZ_PARTY_SITES_N4	2
TABLE ACCESS (BY INDEX ROWID)	HZ_LOCATIONS	2
INDEX (UNIQUE SCAN)	HZ_LOCATIONS_U1	1

3.2. att. Izpildes plāns meklēšanai pēc līguma

SELECT STATEMENT ()		22
NESTED LOOPS ()		22
NESTED LOOPS ()		20
NESTED LOOPS ()		17
NESTED LOOPS ()		15
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTIES	11
INDEX (RANGE SCAN)	HZ_PARTIES_N1	3
INDEX (RANGE SCAN)	OKC_K_PARTY_ROLES_B_U2	2
TABLE ACCESS (BY INDEX ROWID)	OKC_K_HEADERS_B	2
INDEX (UNIQUE SCAN)	OKC_K_HEADERS_B_U1	1
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTY_SITES	3
INDEX (RANGE SCAN)	HZ_PARTY_SITES_N4	2
TABLE ACCESS (BY INDEX ROWID)	HZ_LOCATIONS	2
INDEX (UNIQUE SCAN)	HZ_LOCATIONS_U1	1

3.3 att. Izpildes plāns meklēšanai pēc uzņēmuma nosaukuma

3.3 Meklēšana pēc citiem parametriem

Lai meklēšanu varētu veikt pēc citiem parametriem bija nepieciešams pievienot tabulas kurās šie parametri glabājas. Pēc šādas metodes salīdzinoši vienkārši, papildinot sākuma tabulas ar vienu papildus tabulu, bija izveidot meklēšanu pēc pakalpojuma pieprasījuma numura, klienta konta un rēķina, jo šīs tabulas varēja sasaistīt ar kādu no jau izmantojamām tabulām, neizmantojot citas tabulas. Papildus tabulas ER modelī ir ārpus sarkanā rāmīša. Attiecīgi izpildes plāni mainījās minimāli, jo sakot meklēt šajās tabulās tiek atrasts vai nu līguma id(meklējot pēc rēķina) ar kuru var atrast atbilstošo ieraktu okc_k_headers_b tabulā un tālāk vaicājums izpildās kā 3.2 attēlā redzamajā izpildes plānā, vai arī klienta id, pēc kura var atrast ierakstu hz_parties tabulā un plāns izpildās kā 3.3 attēlā(meklējot pēc konta). Pakalpojuma pieprasījuma gadījuma tiek izmantota pieeja vēl vienam indeksam bez tabulas izmantošanas , lai salīdzinātu vides mainīgos. Pirms labojumu veikšanas meklēt pēc pakalpojuma pieprasījuma nebija iespējams, jo skati pakalpojuma pieprasījuma vietā atgrieza NULL un tabula, kurā varētu tos meklēt netika pievienota. Vaicājumam, kurā tiek meklēts pēc pakalpojuma pieprasījuma, izpildes plāns ir redzams zemāk:

SELECT STATEMENT ()		19
NESTED LOOPS ()		19
NESTED LOOPS ()		17
NESTED LOOPS ()		14
NESTED LOOPS ()		14
NESTED LOOPS ()		12
NESTED LOOPS ()		8
TABLE ACCESS (BY INDEX ROWID)	CS_INCIDENTS_ALL_B	4
INDEX (RANGE SCAN)	CS_INCIDENTS_U2	2
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTIES	2
INDEX (UNIQUE SCAN)	HZ_PARTIES_U1	1
INDEX (RANGE SCAN)	OKC_K_PARTY_ROLES_B_U2	2
TABLE ACCESS (BY INDEX ROWID)	OKC_K_HEADERS_B	2
INDEX (UNIQUE SCAN)	OKC_K_HEADERS_B_U1	1
INDEX (UNIQUE SCAN)	CS_INCIDENTS_ALL_TL_PK	
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTY_SITES	3
INDEX (RANGE SCAN)	HZ_PARTY_SITES_N4	2
TABLE ACCESS (BY INDEX ROWID)	HZ_LOCATIONS	2
INDEX (UNIQUE SCAN)	HZ_LOCATIONS_U1	1

3.4. att. **Izpildes plāns meklēšanai pēc pakalpojuma pieprasījuma numura**

Meklēšana pēc skaitītāja numura un transformatora punkta numura bija nedaudz sarežģītāk, jo atbilstošajām tabulām nav tiešas saiknes ar jau izmantojamajām tabulām, jo tie ir objekta parametri. Tādēļ šajos gadījumos tika veidoti apakšvaicājumi, kuri atgriež divas kolonnas, vienu, kas satur meklējamo nosacījumu un otru, kurā līguma id, kurā šis objekts ir iekļauts. Šī apakšvaicājumam, kas meklē pēc transformatora punkta numura, izmantotās tabulas ir iekļautas zilajā rāmītī.

Gadījumos, ja tiek meklēts objekts vai izmantot saiti, kas saista hz_parties un csi_item_instances, bet, ja tiek meklēti klienti, tad iegūtie ieraksti no apakšvaicājuma ir jāsasaista ar okc_k_lines_b un kns_ctr_counter_usage_items, lai iegūtu līguma identifikatoru, jo tā ir vienīgā drošā sasaiste starp klientu un tam piederošo objektu, jo līgumā ir kvalitātes pārbaude, lai šis objekts nepiederētu diviem klientiem vienlaicīgi. Otra saite starp objektu un tā klientu ER modelī nav attēlota, jo tā ir brīvi ievadāma, parasti, ja skatās šis saites aktīvos datumus, tā ir korekta, taču salīdzinoši bieži ir gadījumi, kad objekts ir sasaistīts arī ar vairāk kā vienu klientu. Meklēšanā pēc transformatora punkta bija jāiekļauj lielākā tabula šajā meklēšanā (17'294'168 ieraksti). Tomēr meklēšana šajā tabulā ir ātra, jo atribūta vērtība(transformatora punkta numurs) un atribūta nosaukums ir indeksēts vienā indeksā, pieeja tabulai nepieciešama tikai lai noskaidrotu tīkla elementa id, kuru tālāk izmantojot var iegūt līgumu, kurā šis objekts ir iekļauts.

NESTED LOOPS ()			36
NESTED LOOPS ()			34
NESTED LOOPS ()			31
NESTED LOOPS ()			30
NESTED LOOPS ()			24
NESTED LOOPS ()			22
NESTED LOOPS ()			20
NESTED LOOPS ()			11
NESTED LOOPS ()			9
TABLE ACCESS (BY INDEX ROWID)	CSI_IEA_VALUES		5
INDEX (RANGE SCAN)	KNS_CSI_IEA_VALUES_N04	"IEV"."ATTRIBUTE_VALUE" LIKE :Z AND "IEV"."ATTRIBUTE1"='KANS_TRANSF_POINT'	4
TABLE ACCESS (BY INDEX ROWID)	CSI_II_RELATIONSHIPS		4
INDEX (RANGE SCAN)	CSI_II_RELATIONSHIPS_N01	"IEV"."INSTANCE_ID"="REL"."OBJECT_ID"	2
TABLE ACCESS (BY INDEX ROWID)	CSI_ITEM_INSTANCES		2
INDEX (UNIQUE SCAN)	CSI_ITEM_INSTANCES_U01	"REL"."SUBJECT_ID"="CII"."INSTANCE_ID"	1
TABLE ACCESS (BY INDEX ROWID)	OKC_K_ITEMS		9
INDEX (RANGE SCAN)	OKC_K_ITEMS_N2	"ITM"."OBJECT1_ID1"=TO_CHAR("CII"."OWNER_PARTY_ID") AND "ITM"."JTOT_OBJECT1_CODE"= 'KNS_OKX_OBJECT'	2
TABLE ACCESS (BY INDEX ROWID)	OKC_K_LINES_B		2
INDEX (UNIQUE SCAN)	OKC_K_LINES_B_U1	"ITM"."CLE_ID"="LINE"."ID"	1
TABLE ACCESS (BY INDEX ROWID)	OKC_K_HEADERS_B		2
INDEX (UNIQUE SCAN)	OKC_K_HEADERS_B_U1	"LINE"."DNZ_CHR_ID"="OKH"."ID"	1
TABLE ACCESS (BY INDEX ROWID)	OKC_K_PARTY_ROLES_B		6
INDEX (RANGE SCAN)	KNS_OKC_K_PARTY_ROLES_B_N6	"OKP"."CHR_ID"="OKH"."ID"	2
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTIES		1
INDEX (UNIQUE SCAN)	HZ_PARTIES_U3	"OKP"."OBJECT1_ID1"=TO_CHAR ("HP"."PARTY_ID")	
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTY_SITES		3
INDEX (RANGE SCAN)	HZ_PARTY_SITES_N4	"PST"."PARTY_ID"="HP"."PARTY_ID" AND "PST"."IDENTIFYING_ADDRESS_FLAG"='Y'	2
TABLE ACCESS (BY INDEX ROWID)	HZ_LOCATIONS		2
INDEX (UNIQUE SCAN)	HZ_LOCATIONS_U1	"LOC"."LOCATION_ID"="PST"."LOCATION_ID"	1

3.5 tabula Izpildes plāns meklēšanai pēc transformatora punkta numura

Gadījumos kad tiek meklēti objekti lielākās tiek izmantota cita starptabula, kas saista objekts ar līgumu, kurai tāpat kā klientu meklēšanas zaros, datu tipi nesakrīt ar paša objekta id datu tipu, tādēļ tiek izmantots tāds pats risinājums kā iepriekš ar okc_k_party_roles_b. Lielākās problēmas tieši objektu meklēšanā sagādāja tas, ka objektu nosaukumos nevalda tik liela dažādība kā uzņēmumu nosaukumos. Kā jau iepriekš tika minēts hz_parties tabulā kopā ir aptuveni 3.3 miljoni ierakstu, no kuriem 1.1 miljons ir objekti. Jau iepriekš uzlabojot citu meklēšanas formu biju izstrādājis risinājumu, ka gadījumos ja objekta nosaukums nav pietiekoši ierobežojošs, tad izpildes plāns tiek mainīts tā, lai pirmajai atlasei tiktu izmantoti citi parametri. Tas tiek darīts ar norāžu palīdzību, jo izmantojot BIND mainīgos nav iespējams izmantot histogrammas. To lietošana būtu arī apgrūtinoša, jo neviens no histogrammu veidiem šai situācijai nav īsti derīgs. Pirmais veids atkrīt, jo dažādo ierakstu skaits ir ļoti liels, jo tajā pašā kolonnā tiek glabāts ne tikai objekta nosaukums, bet arī uzņēmuma nosaukums un klienta vārds un uzvārds. Otrs veids atkrīt, jo iekļaujošais intervāls būtu ļoti garš, lai diviem intervāliem nebūtu vienādas beigu vērtības, vai arī būtu daudz intervālu, kas viens no otra neatšķirtos. Tādēļ datu bāzē tika uztaisīta jauna tabula, kurā tiek glabāti populārākie objektu nosaukumi un to skaits, jo no visiem objektiem aptuveni 45% ir objekts ar nosaukumu "DZĪVOKLIS". Pēc tam, kad visi parametri vaicājumam ir pievienoti, tiek

pārbaudīts cik objektu atbilst ievadītajam objekta nosaukumam. Tas notiek ātri, jo tabulā, kurā glabājas šie dati, ir tikai 54 ieraksti ar objektu nosaukumiem, kuri sistēmā ir vairāk kā 500 ar šo nosaukumu. Kā arī tabulas ieraksts ir īss, jo satur 2 kolonnas – objekta nosaukums un skaits. Pieņemot, ka lietotājs meklē pēc "DZĪVOKLIS%", tiek summēts skaits visiem dzīvokļiem, dzīvokļiem daudzdzīvokļu mājās, dzīvoklis ciematā un citi, kas atbilst meklēšanas kritērijam. Gadījumā, ja šis skaits nepasniedz 5000, tad izmaiņas plānā netiek veiktas. Savukārt, ja šis skaits ir lielāks, tad tiek skatīti, kādi parametri vēl ir ievadīti, lai tos izmantotu sākumā, piemēram, skatītāja numurs, līguma numurs, adrese. Tas tiek izdarīts ar norādi LEADING. Ja par objektam ir ievadīts objekta numurs sistēmā, plānā izmaiņas netiek veiktas, jo objekta numuru Oracle uzskata par labāku kritēriju atlasei un ar to sāk, jo tas ir unikāls.

3.4 Meklēšana pēc adreses

Izmantojot šīs sākuma tabulas, var meklēt arī pēc adreses. Katra adreses daļa tiek ievadīta savā laukā un tiek validēta pēc adrešu reģistra datiem, piemēram, nav iespējams ievadīt kādu adresi Alūksnē, ja sākumā ir ievadīts Rīgas rajons. Validācija nenotiek vienīgi mājas numuram vai nosaukumam. Katrs no šiem laukiem adrešu tabulā ir indeksēts. Šādā situācija meklēšanas rezultāti bija apmierinoši, ja vien netika ievadīts neprecīzs mājas numurs, t.i. tās satur "%" tuvu virknes sākuma, vai pat virkne ar to sākās. Jo pēc izpildes plāna, gadījumos, kad tika ievadīta gan iela, gan mājas numurs, tabulā ieraksti tika meklēti caur mājas numura indeksu, bet iela kalpoja kā papildus atlasošais parametrs. Sliktākais iespējams parametrs bija "%1%", jo šādam nosacījuma atbilst aptuveni 500'000 adreses no 2'407'620 adresēm. Tajā pašā laikā "sliktākā" ielā ir Brīvības iela ar 33'000 adresēm. Vēl tuvu 30'000 robežai ir Maskavas, Skolas un Rīgas ielas. Veicot skatīšanu sistēmā uzzināju, ka vispār ir 70 ielas kurām atbilst vairāk kā 5000 adreses(neņemot vērā pilsētu). Tādēļ tika izveidots indekss ielai un mājas numuram kopā, kas atrisināja šo problēmu arī citās formās, kur notika meklēšana pēc adreses. Sākumā tikai objektu, bet vēlāk arī pārējiem meklēšanas zariem, tikai izveidots speciālgadījums, ja tiek ievadīts mājas nosaukums ar % pirmajās 3 simbolu vietās un nav ievadīta iela. Šādā gadījumā, lai atrastu adreses id, kuru varētu tālāk izmantot, tiek izmantoti divi indeksi. Viens no tiem satur visus mājas numurus/nosaukumus un otrs adrešu id. Šie indeksi tiek sasaistīti ar ROWID un ar norādes INDEX_FFS palīdzību tiek panāks, lai visas adrešu id, kas atbilst ievadītajam mājas nosaukumam tiktu atlasīti uzreiz, nevis kā parasti atrod vienu ierakstu, tad griežas pie pārējam tabulām un pārbauda pārejos nosacījumus. Pirms risinājuma realizācijas tika veikta adrešu skatīšana grupējot pēc pirmajiem simboliem. Tās rezultātā tika secināts, ka ir zināmi pirmie 4 simboli precīzi, tad atgriežamo ierakstu skaits būs mazāks par 5000,

kas ir maz no visas tabulas ierakstiem. Šie iegūtie adrešu id tālāk tiek apstrādāti kā parasti. Šeit ir izpildes plāns, ka izskatās, ja meklēšana notiek pa šādu zaru:

3.6. att. Izpildes plāns objektu meklēšanai pēc neprecīza mājas numura

SELECT STATEMENT ()				2312
NESTED LOOPS ()				2312
NESTED LOOPS ()				2310
NESTED LOOPS ()				2308
NESTED LOOPS ()				2218
NESTED LOOPS ()				2188
NESTED LOOPS ()				2101
HASH JOIN ()		"LOC2".ROWID="LOC1".ROWID		2043
INDEX (FAST FULL SCAN)	HZ_LOCATIONS_N24		UPPER("LOC1"."STREET_NUMBER") LIKE UPPER(:Z)	1223
INDEX (FAST FULL SCAN)	HZ_LOCATIONS_U1			816
TABLE ACCESS (BY INDEX ROWID)	HZ_LOCATIONS			2
INDEX (UNIQUE SCAN)	HZ_LOCATIONS_U1	"LOC2"."LOCATION_ID"="LOC"."LOCATION_ID"		1
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTY_SITES		"PST"."IDENTIFYING_ADDRESS_FLAG"='Y'	3
INDEX (RANGE SCAN)	HZ_PARTY_SITES_N2	"LOC"."LOCATION_ID"="PST"."LOCATION_ID"		2
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTIES		"HP"."ATTRIBUTE_CATEGORY"='OBJECT'	2
INDEX (UNIQUE SCAN)	HZ_PARTIES_U1	"PST"."PARTY_ID"="HP"."PARTY_ID"		1
TABLE ACCESS (BY INDEX ROWID)	OKC_K_ITEMS			9
INDEX (RANGE SCAN)	OKC_K_ITEMS_N2	"ITM"."OBJECT1_ID1"=TO_CHAR("HP"."PARTY_ID") AND "ITM"."JTOT_OBJECT1_CODE"='KNS_OKX_OBJECT'	"HP"."PARTY_ID"=TO_NUMBER("ITM"."OBJECT1_ID1")	2
TABLE ACCESS (BY INDEX ROWID)	OKC_K_LINES_B		NVL("LINE"."START_DATE", TRUNC(SYSDATE))<= TRUNC(SYSDATE) AND NVL("LINE"."END_DATE", TRUNC(SYSDATE))>= TRUNC(SYSDATE)	2
INDEX (UNIQUE SCAN)	OKC_K_LINES_B_U1	"ITM"."CLE_ID"="LINE"."ID"		1
TABLE ACCESS (BY INDEX ROWID)	OKC_K_HEADERS_B		"OKH"."STS_CODE"='ACTIVE' OR "OKH"."STS_CODE"='QA_HOLD'	2
INDEX (UNIQUE SCAN)	OKC_K_HEADERS_B_U1	"LINE"."DNZ_CHR_ID"="OKH"."ID"		1

SECINĀJUMI UN REZULTĀTI

Risinājums uz darba rakstīšanas brīdi vēl nav nonācis produkcijā, kaut arī veiktspējas un datu pareizības problēmas ir atrisinātas, jo klienta pieteikuma risinājuma ietvaros ir nepieciešams arī mainīt formas vizuālo izskatu un pievienot jaunos meklēšanas parametrus. Kā arī labot to, ka forma neatļauj meklēt pēc rēķina numura. Taču darbā izstrādātais risinājums jau izstrādājot tika veidots tā, lai visas procedūras var izsaukt arī no vecās formas versijas. Tālāk izklāstītie rezultāti ir no testa vides, kas ir produkcijas kopija ar praktiski to pašu datu apjomu(datus nav datu, kas nākuši pēdējo pāris mēnešu laikā). Kaut arī dati produkcijā ir nedaudz vairāk, rezultāti tiktu iegūti mazliet ātrāk dēļ aparatūras atšķirībām.

Testēšanas nolūkos testa vidē tika izveidots jauns klients. Lielākā laika starpība, kas tika iegūta veicot formas izmaiņas ir meklēšanā pēc skaitītāja numura. Pirms labojumu veikšanas nebija iespējams iegūt rezultātus 20 minūšu laikā. Pielikumā ir pievienots izpildes plāns, kur ar sarkanu ir atzīmēta sliktā vaicājuma vieta(bez nosacījumiem tiek pievienots aptuveni 1 miljons ierakstu, no kuriem vēlāk tiek atfiltrēti liekie). Pēc labojumu uzstādīšanas rezultāti tika iegūti ātrāk par sekundi. Tālāk doti piemēri citām meklēšanām:

pēc nosaukuma “HAIMANS SIA” - 92 sekundes

pēc Latvijas Uzņēmumu reģistra numura “10010010010” – 88 sekundes

pēc līguma numura 1004533242 – 61 sekunde

pēc adreses “Taku iela 12, Rēzeknes rajons” - 4 minūtes

pēc vārda “HAIMS”un uzvārda “HAIMANS” - 87 sekundes.

Meklējot objektu visi laiki bija līdzīgi izņemot, ja meklēja pēc līguma numura, kur rezultāta iegūšanai vajadzēja 2 sekundes, jo izpildes plāns bija ļoti tuvs mērķim, kādu es gribēju sasniegt tikai šajā gadījumā papildus bija iesaistītas liekas tabulas. Pēc labojumu uzstādīšanas meklēšana aizņēma mazāk kā 1 sekundi. Meklēšana pēc adreses un mājas numura dēļ indeksa izveides uzlabojās visās formās, kur tā tika izmantota, it sevišķi gadījumos, ja mājas numurs ir ar “%“ sākumā. Meklēšana tikai pēc mājas nosaukuma ar “%” pirmo 4 simbolu vietā aizņem aptuveni 2,5 minūtes, praktiski neatkarīgi no ievadītajām vērtībām, jo tiek izmantota indeksu pilna ielasīšana. Iespējams, šo problēmu var novērst papildināt šīs adrešu zara meklēšanu ar līdzīgu slikto simbolu sarakstu, kā objektiem, lai izvairītos no nevēlamajām kombinācijām, un gadījumā, ja simbolu

virkne ir pietiekoši unikāla, meklēt pēc indeksa kā parasti.

Nākotnē izveidoto risinājumu, iespējams, varētu vēl uzlabot pārskatot visu iesaistīto tabulu indeksus, piemēram, tabulai csi_iea_values jau izmantotā indeksa vietā izveidot citu, kas iekļauj arī instance_id, tādējādi izvairoties no pašas tabulas izmantošanas. Vai arī citām tabulā pie ievadītiem "sliktiem" parametriem, izveidot savādāku izpildes plānu, kā tika darīt ar meklēšanu pēc mājas numura ar "%" sākumā. Šo pašu metodi ar vairāku indeksu apvienošanu, iespējams, var izmantot arī citām tabulām. Šajā darbā izveidotais risinājums ir izmantojams arī jau ievadā minētajai tehnisko darbu meklēšanas formai, jo, kaut arī tai nav tik lielas problēmas ar veiktspēju, kā bija šai formai, tomēr noteikti var uzlabot, jo šobrīd šī forma, lai iegūta gala rezultātu izmanto četras starptabulas. Lai darbā izveidoto risinājumu varētu pielietot arī šeit, tai būtu nepieciešams mazliet pamainīt sākotnējo kodolu, jo tiek prasīti citi dati(meklēšanas kategorijas ir skaitītājs, objekts, līgums) un papildināt kritēriju izvēli. Šajā risinājumā izmantotie kritēriji paliktu, kādi ir, jo meklēšana ir iespējama arī pēc tiem, kā arī lielāko meklēšanu daļu pēc tehniskajiem datiem var iegūt pamainot tikai vienu nosacījumu meklēšanai pēc transformatora punkta numura (ER modelī zilajā rāmītī iekļautais apakšvaicājums). Šī risinājuma ieviešanas iespējas Universālās meklēšanas formā ir ierobežotas, jo tajā par pamatu tiek izmantot jau gatavi skati, lai mainītu skatus, bija nepieciešams labot formas standarta bibliotēku.

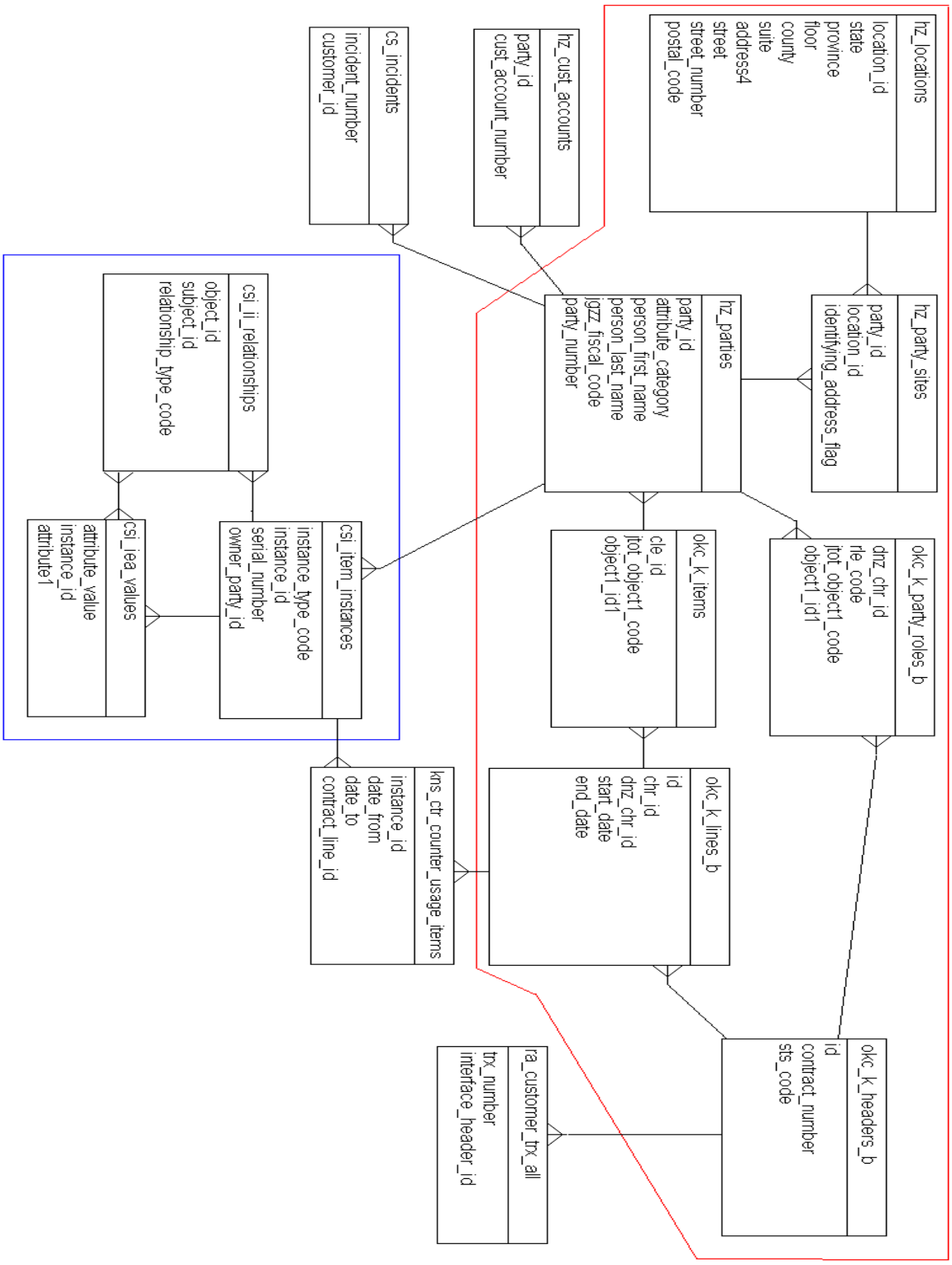
IZMANTOTĀ LITERATŪRA UN AVOTI

1. **Kost Stephen** *An introduction to SQL Injection Attacks for Oracle Developers* [tiešsaiste]. Chicago: Integrity Corporation, 2004 - [atsauce 20.05.2007]. Pieejams: www.net-security.org/dl/articles /IntegrigyIntrotoSQLInjectionAttacks.pdf
2. **Litchfield David** *Cursor Injection* [tiešsaiste]. Next generation Security Software Ltd, 2007 – [atsauce 22.06.2007]. Pieejams: www.databasesecurity.com/dbsec/cursor-injection.pdf
3. **Tow Dan** *SQL Tuning* Sebastopol:O'Reilly, 2004. 336 p.
4. **Engsig Bjørn** *Efficient use of bind variables, cursor_sharing and related cursor parameters* [tiešsaiste] Miracle A/S, 2002 - [atsauce 18.05.2007]. Pieejams: www.miracleas.dk/tools/Miracle_2_cursor.pdf
5. *Oracle9i Database Performance Tuning Guide and Reference* [tiešsaiste]. Oracle, 2002 – [atsauce 15.05.2007]. Pieejams:http://www.oracle.com/pls/db92/db92.docindex
6. **Lorentz Diana** *Oracle Database SQL Reference 10g Release 2* [tiešsaiste]. Oracle, 2005 – [atsauce 15.05.2007]. Pieejams:http://docs.huihoo.com/oracle/docs/B19306_01/server.102/b14200/toc.htm
7. **Fogel Steve, Lane Paul** *Oracle Database Administrator's Guide, 10g Release 2* [tiešsaiste]. Oracle, 2005 – [atsauce 15.05.2007]. Pieejams:http://docs.huihoo.com/oracle/docs/B19306_01/server.102/b14231/title.htm
8. **Gurry Mark** *Oracle SQL Tuning Pocket Refrence* Sebastopol:O'Reilly, 2004. 108 p.

PIELIKUMI

1. pielikums - izmantoto tabulu ER modelis

Izmantoto tabulu ER modelis



Izpildes plāns meklēšanai pēc skaitītāja numura

INSERT STATEMENT ()		5019
COUNT (STOPKEY)		
NESTED LOOPS ()		5019
TABLE ACCESS (BY INDEX ROWID)	HZ_LOCATIONS	2
INDEX (UNIQUE SCAN)	HZ_LOCATIONS_U1	1
NESTED LOOPS ()		5017
NESTED LOOPS ()		5014
NESTED LOOPS ()		5012
NESTED LOOPS ()		5009
NESTED LOOPS ()		5006
NESTED LOOPS ()		5004
NESTED LOOPS ()		4996
MERGE JOIN (CARTESIAN)		4994
TABLE ACCESS (BY INDEX ROWID)	CSI_IEA_VALUES	11
NESTED LOOPS ()		23
NESTED LOOPS ()		12
NESTED LOOPS ()		11
NESTED LOOPS ()		8
NESTED LOOPS ()		7
TABLE ACCESS (BY INDEX ROWID)	CSI_ITEM_INSTANCES	4
INDEX (RANGE SCAN)	CSI_ITEM_INSTANCES_N05	3
NESTED LOOPS ()		3
TABLE ACCESS (BY INDEX ROWID)	HR_ALL_ORGANIZATION_UNITS_TL	2
INDEX (RANGE SCAN)	HR_ALL_ORGANIZATION_UNTS_TL_N2	1
TABLE ACCESS (BY INDEX ROWID)	HR_ALL_ORGANIZATION_UNITS	1
INDEX (UNIQUE SCAN)	HR_ORGANIZATION_UNITS_PK	
TABLE ACCESS (BY INDEX ROWID)	CSI_II_RELATIONSHIPS	3
INDEX (RANGE SCAN)	CSI_II_RELATIONSHIPS_N02	2
INDEX (UNIQUE SCAN)	CSI_ITEM_INSTANCES_U01	1
TABLE ACCESS (BY INDEX ROWID)	CSI_II_RELATIONSHIPS	3
INDEX (RANGE SCAN)	CSI_II_RELATIONSHIPS_N02	2
INDEX (UNIQUE SCAN)	CSI_ITEM_INSTANCES_U01	1
INDEX (RANGE SCAN)	CSI_IEA_VALUES_N01	2
BUFFER (SORT)		4983
INDEX (RANGE SCAN)	OKC_K_PARTY_ROLES_B_U2	4971
TABLE ACCESS (BY INDEX ROWID)	OKC_K_HEADERS_B	2
INDEX (UNIQUE SCAN)	OKC_K_HEADERS_B_U1	1
TABLE ACCESS (BY INDEX ROWID)	KNS_BILLS	8
INDEX (RANGE SCAN)	KNS_BILLS_N1	2
TABLE ACCESS (BY INDEX ROWID)	RA_CUSTOMER_TRX_ALL	2
INDEX (UNIQUE SCAN)	RA_CUSTOMER_TRX_U1	1
TABLE ACCESS (BY INDEX ROWID)	OKC_K_LINES_B	3
INDEX (RANGE SCAN)	OKC_K_LINES_B_N1	2
TABLE ACCESS (BY INDEX ROWID)	OKC_K_ITEMS	3
INDEX (RANGE SCAN)	OKC_K_ITEMS_N4	2
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTIES	2
INDEX (UNIQUE SCAN)	HZ_PARTIES_U1	1
TABLE ACCESS (BY INDEX ROWID)	HZ_PARTY_SITES	3
INDEX (RANGE SCAN)	HZ_PARTY_SITES_N4	2

Bakalaura darbs „SQL optimizācija meklēšanas formā” izstrādāts LU Fizikas un matemātikas fakultātē.

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

Autors: Andris Tretjakovs

Rekomendēju darbu aizstāvēšanai

Vadītāja: lektors M. dat. Aivars Niedrītis

Recenzents: docents Dr.dat. Ģirts Karnītis

Darbs iesniegts Datorikas nodaļā 04.06.2007.

Metodiķe: Ārija Sproģe

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

Komisijas sekretāre: