LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SQL DATUBĀŽU OPTIMIZĀCIJAS IESPĒJAS UN SALĪDZINĀJUMS

BAKALAURA DARBS

Autors: Lauma Šivare
Studenta apliecības Nr.: ls11120
Darba vadītājs: docents Dr. sc. comp. Aivars Niedrītis

RĪGA 2015

ANOTĀCIJA

Datubāzes veiktspēja ir ļoti atkarīga no optimizatora darbības, zināmiem datubāzes defektiem un neprecīzas statistikas. Saturīgas statistikas palīdz strādāt un izvēlēties ātrākos SQL pieprasījumu izpildes plānus. Lai optimizators spētu atbilstoši darboties, tam ir nepieciešamas kvalitatīvas statistikas, bet bieži vien to vākšanas process ir laikietilpīgs un resursus patērējošs.

Šajā darbā tiek apskatīti un salīdzināti populārāko datubāžu optimizatoru darbības principi un uzbūve, statistiku vākšanas uzstādījumi un parametri gan teorētiski, gan praktiski. Bakalaura darbs uzskatāms par informatīvu materiālu ar mērķi apkopot informāciju par datubāzēm un optimizācijas iespējām.

Atslēgvārdi – *Oracle 11g* datubāze, *Microsoft SQL Server 2012* datubāze, optimizators, statistikas, pieprasījumu izpildes plāns

ABSTRACT

„SQL database optimization options and comparison”

Database performance is dependent on query optimizer activity, certain database faults and inaccurate statistics. Qualitative statistics help query optimizer to choose the fastest SQL query execution plans. For optimizer to operate accurately, it is necessary to gather high-quality statistics, but often this process is time and resources consuming.

In this work are examined and compared the most popular database optimizer principles and structure, statistics gathering settings and parameters both theoreticaly and practicaly. Thesis is considered to be informational material in order to gather information about databases and optimization capabilities.

Keywords – *Oracle 11g* database, *Microsoft SQL Server 2012* database, query optimizer, optimizer statistics, execution plan

SATURS

APZĪMĒJUMU SARAKSTS 6
IEVADS 7
1. POPULĀRĀKĀS DATUBĀZES 8
1.1. DB-ENGINES 8
1.2. DBMS vispārējs salīdzinājums 9
2. TEORĒTISKĀ DAĻA 11
2.1. Optimizatora statistikas 11
2.1.1. Oracle 11g datubāze 11
2.1.2. Microsoft SQL Server 2012 datubāze 16
2.2. Optimizators 17
2.2.1. Oracle 11g datubāze 17
2.2.1. Microsoft SQL Server 2012 datubāze 22
2.3. Pieprasījuma izpildes plāns 24
2.3.1. Oracle 11g datubāze 24
2.3.2. Microsoft SQL Server 2012 datubāze 25
2.4. Optimizatora parametri 26
2.4.1. Oracle 11g datubāze 26
2.4.2. Microsoft SQL Server 2012 datubāze 28
2.5. Teorijas salīdzinājums 29
3. PRAKTISKĀ DAĻA 30
3.1. Datubāžu uzstādīšana 31
3.1.1. Oracle 11g datubāzes uzstādīšana 31
3.1.2. Microsoft SQL Server 2012 datubāzes uzstādīšana 33
3.2. HR shēma 35
3.3. Datu ievade, statistiku vākšana un izpildes plānu ģenerēšana Oracle un Microsoft SQL Server 2012 datubāzē 38
3.4. Praktiskās daļas rezultātu salīdzinājums 40
3.4.1. Oracle 11g datubāzes rezultāti 40
3.4.2. Microsoft SQL Server 2012 datubāzes rezultāti 42
REZULTĀTI 44
SECINĀJUMI 45
IZMANTOTĀ LITERATŪRA UN AVOTI 46
PIELIKUMI 48
1. pielikums – Relāciju modelis HR shēmai 48
2. pielikums – Tabulu izveide Oracle datubāzē 49

3. pielikums – Tabulu izveide Microsoft SQL Server datubāzē 53
4. pielikums – HR shēmas tabulu apraksts 58
5. pielikums – Sākuma datu ievades komandas 60
6. pielikums – Trīs izmantotie SQL pieprasījumi praktiskajā daļā 61
7. pielikums – Oracle 11g praktiskās daļas rezultāti 62
8. pielikums – Microsoft SQL Server 2012 praktiskās daļas rezultāti 72

APZĪMĒJUMU SARAKSTS

1. DBMS (ang. *Database Management System*) – savstarpēji saistītu informacionālu objektu tematisks kopums, kas ar speciālas pārvaldības sistēmas starpniecību organizēts tā, lai nodrošinātu ērtu informācijas izguvi, izdarītu tās atlasi un kārtošanu.
2. IT (ang. *Information technology*) – zināšanu, metožu, paņēmienu un tehniskā aprīkojuma kopums, kas ar datoru un sakaru līdzekļu starpniecību nodrošina jebkuras informācijas iegūšanu, glabāšanu un izplatīšanu.
3. Izpildes plāns – (ang. *Execution Plan*) – Komandā, pieprasījumā vai programmā paredzēto darbību veikšana.
4. SQL – (ang. *Structured Query Language*) – strukturēta vaicājumvaloda.
5. T-SQL (ang. *Transact Structured Query Language*) – strukturētas vaicājumvalodas paplašinājums.
6. Optimizators – (ang. *Query Optimizer*) – datubāžu pārvaldības sistēmās iebūvēta programmatūra, kas atrod visefektīvāko veidu kā izpildīt SQL pieprasījumus.
7. CPU – centrālais procesors.
8. OUI - (ang. *Oracle Universal Installer*) – programma, kas paredzēta *Oracle* datubāzes uzstādīšanai.
9. SQL Server Installation Center – programma, kas paredzēta *Microsoft SQL Server* datubāzes uzstādīšanai.
10. SGA - (ang. *System Global Area*) - koplietojama atmiņa, kas satur datus un kontroles informāciju vienai *Oracle* instancei.
11. PGA - (ang. *Program Global Area*) - atmiņa, kas satur datus un kontrolē servera procesu informāciju.
12. RAM - (ang. *Random Access Memory*) - atmiņa, kurā datus var gan ierakstīt , gan lasīt no tās.
13. Pārbaude - (ang. *Scan*) – optimizatora SQL izpildes plāna piekļuves ceļa indeksa pārbaude.
14. Sasniegšana - (ang. *Seek*) - optimizatora SQL izpildes plāna piekļuves ceļa indeksa sasniegšana.
15. B koks - (ang. *B-Tree*) - Datu veidojums, kas satur hierarhiski savā starpā saistītus mezglus ar ne vairāk kā vienu virsmezglu katram mezglam un tikai vienu saknes mezglu.
16. Mezgls (ang. *cluster*) – Vienādu vai līdzīgu indeksu grupa datubāzē, kas atrodas kopā vai ir kopsaistīti.

IEVADS

Datubāzes kvalitatīva veiktspēja ir atkarīga no tādiem lielumiem kā tīkla uzstādījumiem, datubāzes parametru konfigurācijas, datubāzes noslodzes utt. Kā vienu no galvenajiem veiktspējas rādītājiem var minēt optimizatora darbību, kas automātiski izvēlas ātrākos un efektīvākos SQL pieprasījumu izpildes plānus un izmanto statistiku vākšanas tehnoloģijas.

Optimizatora statistikas palīdz strādāt un izvēlēties ātrākos izpildes plānus optimizatoram. Lai tas spētu atbilstoši darboties, tam ir nepieciešamas kvalitatīvas statistikas, bet bieži vien to vākšanas process ir laikietipīgs un resursus patērējošs.

Darbā pētāmā problēma – jāapzin un jāsalīdzina populārāko datubāžu (*Oracle, Microsoft SQL Server*) optimizācijas iespējas, optimizatora darbība, veidi, kā tiek izvēlēts konkrēts izpildes plāns.

Darba mērķi:

1. Apzināt dažādās optimizācijas iespējas populārākajās datubāzēs.
2. Salīdzināt datubāžu optimizācijas iespējas pēc izpildes ātruma, parametriem un kvalitātes.
3. Izveidot informatīvu materiālu, kas aprakstīs datubāžu optimizācijas iespējas, palīdzēs vieglāk izvēlēties piemērotāko datubāzi savām vajadzībām un dos vispārēju salīdzinājumu.

Darbs sastāv no apzīmējumu saraksta, ievada, teorētiskās daļas ar apakšnodaļām, kurās ir aprakstīta informācija par *Oracle* un *Microsoft SQL Server* datubāžu optimizatioru, tā statistikām un parametriem, pieprasījuma izpildes plāniem, un raktiskās daļas ar apakšnodaļām, kurās ir pētīts *Oracle* un *Microsoft SQL Server* optimizācijas process pirms un pēc datu ievadēm tabulās un optimizatora statistiku vākšanas, rezultātiem, secinājumiem, izmantotās literatūras saraksta un astoņiem pielikumiem.

1. POPULĀRĀKĀS DATUBĀZES

Veidojot populārāko datubāžu salīdzinājumu, ir nepieciešams noskaidrot izplatītākās datubāzes un to aprakstus. Pamatā tiek ņemta DB-ENGINES tīmekļa vietne, kas tiek atjaunināta katru mēnesi ar jaunāko informāciju un grafikiem.

1.1. DB-ENGINES

DB-ENGINES veido datubāžu pārvaldības sistēmu rangu, pamatojoties uz DBMS popularitāti, kas tiek mērīta pēc sekojošiem parametriem: [1]

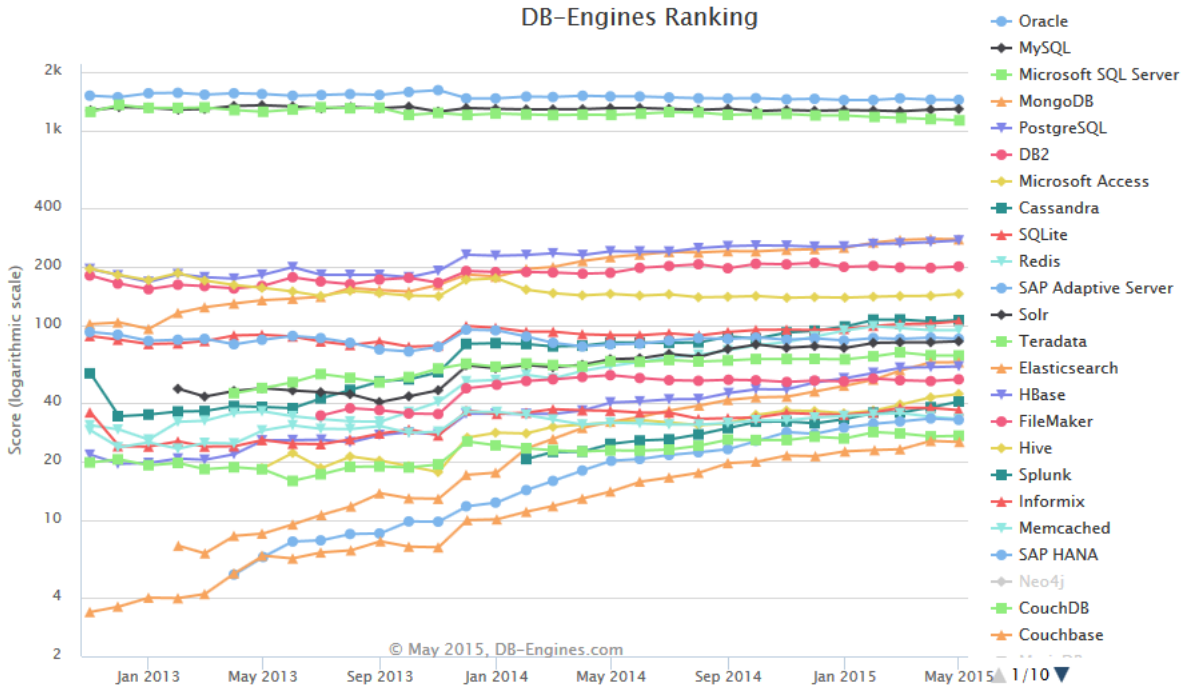
1) sistēmu pieminēšanas skaits dažādās tīmekļa vietnēs, izmantojot tādu meklētājprogrammu kā *Google* un *Bing* vaicājumu skaitu;
2) vispārēja interese par sistēmu, izmantojot meklēšanas biežumu rīkā *Google Trends*;
3) sistēmu tehnisko stāvokļu apspriešanas biežums tādās diskusiju tīmekļa vietnēs, kas ir cieši saistītas ar IT nozari, kā *Stack Owerflow* un *DBA Stack Exchange*;
4) sistēmu pieminēšanas biežums personu prasmju aprakstos *LinkedIn*;
5) sistēmu izplatība sociālajos tīklos un ziņās, izmantojot *Twitter*.

Izpētot DB-ENGINES tīmekļa vietnes saturu, tika noskaidrots, ka populārākās datubāzes gan 2014. gada maijā, gan 2015. gada maijā ir *Oracle*, *MySQL* un *Microsoft SQL Server*. [2] (skat. 1.1. tabulu) Citas DBMS datubāzes šajā darbā netiek apskatītas.

1.1. tabula – **DB-ENGINES rangs 2015. gada maijā** [2]

Rangs			DBMS	Datubāžu modelis
Maijs, 2015	Aprīlis, 2015	Maijs, 2014		
1.	1.	1.	*Oracle*	Relāciju DBMS, komerciāla
2.	2.	2.	*MySQL*	Relāciju DBMS, atvērtā pirmkoda
3.	3.	3.	*Microsoft SQL Server*	Relāciju DBMS, komerciāla

Ir iespējams arī iegūt vispārēju informāciju par šo trīs datubāžu popularitāti, sākot ar 2013. gada janvāri. [3] (skat. 1.1. att.)



1.1. att. – **DB-ENGINES rangs no 2013. gada janvāra līdz 2015. gada maijam** [3]

Kā redzams 1.1. attēlā *Oracle*, *MySQL* un *Microsoft SQL Server* jau pāris gadus ir līderi tirgū popularitātes ziņā, apsteidzot citas datubāžu pārvaldības sistēmas.

1.2. DBMS vispārējs salīdzinājums

Tabulā 1.2. attēlots vispārējs *Oracle* un *Microsoft SQL Server* salīdzinājums. [4] Darbā netiek apskatīta *MySQL* datubāze, jo tā kopš 2014. gada ir *Oracle* produkts.

1.2. tabula – **Vispārējs *Oracle* un *Microsoft SQL Server* salīdzinājums** [4]

Nosaukums	*Oracle*	*Microsoft SQL Server*
Apraksts	Plaši izmantota relāciju DBMS	*Microsoft* relāciju DBMS
Izstrādātājs	*Oracle*	*Microsoft*
Sākotnējās versijas izdošanas gads	1980	1989
Pašreizējā versija	12 Release 1 (12.1.0.2), 2014. gada jūlijs	SQL Server 2014, 2014. gada aprīlis
Licence	Komerciāla	Komerciāla
Servera operētājsistēmu atbalsts	*AIX*, *HP-UX*, *Linux*, *OS X*, *Solaris*, *Windows*	*Windows*
SQL	Jā	Jā

Atbalstītās programmēšanas valodas	*C, C#, C++, Clojure, Cobol, Eiffel, Erlang, Fortran, Groovy, Haskell, Java, JavaScript, Lisp, Objective C, Perl, PHP, Python, R, Ruby, Scala, Tcl, Visual Basic*	*.Net* *Java* *PHP* *Python* *Ruby* *Visual Basic*

Pamatojoties uz vispārējo *Oracle* un *Microsoft SQL Server* salīdzinājumu, praktiskajā daļā tiks izmantotas sistēmu versijas, kas ir vecākas par 2014. gadu. Kā stabilas un darba tirgū izplatītas DBMS versijas ir uzskatāmas *Oracle 11g* un *Microsoft SQL Server 2012*. [18]

2. TEORĒTISKĀ DAĻA

Teorētiskajā daļā tiek apskatītas *Oracle 11g* un *Microsoft SQL Server 2012* datubāžu optimizācijas iespējas un parametri.

2.1. Optimizatora statistikas

Statisktikas parāda vispārīgas detaļas par datubāzi un tās objektiem, kuras izmanto optimizators, lai izvēlētos katra SQL pieprasījuma izpildes plānu. Tās tiek glabātas datu vārdnīcā un ir pieejamas, izmantojot datubāzes skatus. [5]

Optimizatora statistikas satur informāciju par tabulām, rindu un bloku skaitu tajās un katras rindas vidējo garumu. Tiek ievākta arī informācija par katras tabulas kolonnām, unikālo vērtību skaitu tajās un histogrammām, ja tādas tiek lietotas. Optimizatora statistikas satur arī informāciju par indeksiem un sistēmas uzstādījumiem. [5]

2.1.1. *Oracle 11g* datubāze

Oracle datubāzes ātrdarbība ir lielā mērā atkarīga no *Oracle* optimizatora darbības. Bet, lai tas spētu darboties korektāk, ir nepieciešams savākt saturīgas un pēc iespējas pilnīgākas datubāzes statistikas. Šīm statistikām datu vārdnīcā un sistēmā ir jābūt pēc iespējas akurātākām un jāatspoguļo apjomu un datu sadalījumu tabulās, indeksos un shēmās. Ir nepieciešams statistikas vākt periodiski.

Vecākās *Oracle* versijās tiek lietota DBMS_STATS procedūra, lai savāktu statistikas, bet jaunākās versijās izmanto FND_STATS, kas savas darbības laikā, izmanto arī DBMS_STATS procedūru. [18]

Statistikas var vākt visai datubāzei, fiksētiem objektiem, tabulām, shēmām, kolonnām vai konkrēti atsevišķiem objektiem ar dažādiem parametriem. Statistiku vākšanas procesu raksturo katra uzņēmuma ideoloģija un darbinieku, atbildīgo personu uzskati un vēlmēs.

Optimizatora statistikas ir iespējams vākt gan automātiski, gan manuāli. [18]

DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC procedūra ir *Oracle* ieteikta automātiskas statistikas vākšanas gadījumam. [5] Ir iespējams izvēlēties statistiku vākšanas biežumu un dažādus parametrus, kas iegūst informāciju par visu datubāzi kopumā vai arī tikai par atsevišķiem objektiem. Izvēle, vai atstāt ieteicamos uzstādījumus, vai mainīt tos atbilstoši sistēmas vajadzībām, ir atkarīga no datubāžu administratoriem. [5]

Optimizatora statistiku manuāla vākšana arī tiek veikta ar DBMS_STATS pakotni, mainot parametrus pēc nepieciešamības un ieskatiem. Atjauninot statistikas datu vārdnīcā, iepriekšējās tiek saglabātas, lai nepieciešamības gadījumā būtu iespēja atjaunot iepriekšējās

statistikas. Ir iespējamas dažādas DBMS_STATS pakotnes procedūras un parametri, kas veido dažāda satura un apjoma datus par datubāzi. Vācot statistikas, ir iespējams nenorādīt procedūras parametrus. Tādā gadījumā *Oracle* izvēlas parametru noklusētās vērtības. [5]

DBMS_STATS pakotnes procedūrām ir iespējami dažādi parametri. To pielietojums ir atšķirīgs, un ir noderīgi zināt, ko katrs parametrs dara. (skat. 2.1. tabulu) [5,6]

2.1. tabula – **DBMS_STATS pakotnes procedūru parametru apraksti [5,6]**

Parametrs	Apraksts
ESTIMATE_PERCENT	Procentuālais rindu novērtējums. Vērtības ir iespējamas no 0.000001 līdz 100. Ir ieteicams izmantot procedūru DBMS_STATS.AUTO_SAMPLE_SIZE, kas automātiski izvēlas procentuālo rindu novērtējumu katrai tabulai, ņemot vērā dažādus operētājsistēmas un datubāzes uzstādījumus.
BLOCK_SAMPLE	Norāda, vai statistiku vākšanā izmantot nejaušu bloku vai rindu iztveršanu.
METHOD_OPT	Norāda, vai tiks vāktas arī kolonnu histogrammas.
DEGREE	Statistiku vākšanas paralēlisma pakāpe. Ir atkarīgs no sistēmā norādīto CPU skaita.
CASCADE	Norāda, vai tiek vāktas indeksu statistikas. Šis parametrs ir pielīdzināms DBMS_STATS.GATHER_INDEX_STATS.
STATTAB	Norāde uz tabulu, kurā tiks glabātas pašreizējās statistikas.
STATID	STATTAB parametra unikāls identifikators.
OPTIONS	Norāda, kā un kādiem objektiem vākt statistikas: - GATHER – visiem objektiem; - GATHER AUTO – *Oracle* novērtē, kuriem objektiem ir nepieciešams savākt statistikas un to izdara automātiski; - GATHER STALE – sen neizmantotiem objektiem, pamatojoties uz *_TAB_MODIFICATIONS skatiem; - GATHER EMPTY – objektiem, kuriem

	pašlaik nav vāktas statistikas; - LIST AUTO – atgriež sarakstu ar objektiem, kas tiktu apstrādāti ar GATHER AUTO parametru; - LIST STALE - atgriež sarakstu ar objektiem, kas tiktu apstrādāti ar GATHER STALE parametru; - LIST EMPTY - atgriež sarakstu ar objektiem, kas tiktu apstrādāti ar GATHER EMPTY parametru;
OBJLIST	Saraksts ar objektiem, kam nevākt statistikas un kas ir uzskatāmi par tukšiem vai neizmantotiem.
STATOWN	STATTAB parametra shēmas nosaukums.
GATHER_SYS	Norāda, vai vākt statistikas objektiem, kas pieder SYS jeb galvenajam sistēmas lietotājam.
GRANULARITY	Nosaka statistiku vākšanas detalizāciju. Attiecas tikai uz tabulām, kas ir sadalītas nodalījumos.
NO_INVALIDATE	Ja parametra vērtība norādīta kā TRUE, atkarīgie kursori netiek padarīti par nederīgiem.
OBJ_FILTER_LIST	Norāda statistiku vākšanā iesaistīto objektu filtru.
COMP_ID	Analizējamās shēmas sastāvdaļas identifikators.
OWNNAME	Analizējamās tabulas shēmas nosaukums.
INDNAME	Indeksa nosaukums.
PARTNAME	Partīcijas nosaukums.
FORCE	Norāda, vai vākt statistikas tabulai, ja arī tā sistēmā ir uzrādīta kā aizturēta.
TABNAME	Tabulas nosaukums.
STATTYPE	Statistiku tips. Vienīgā vērtība ir DATA.
GATHERING_MODE	Iespējami 4 dažādi sistēmas statistiku vākšanas parametri – NOWORKLOAD, INTERVAL, START, STOP.
INTERVAL	Laiks minūtēs, kad vākt statistikas. Var norādīt tikai tad, ja GATHERING_MODE=’INTERVAL’.

DBMS_STATS.GATHER_DATABASE_STATS procedūra tiek lietota gadījumos, ja ir nepieciešams savākt optimizatora statistikas visiem objektiem, tabulām un indeksiem datubāzē. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

DBMS_STATS.GATHER_DICTIONARY_STATS procedūra tiek lietota gadījumos, kad ir nepieciešams savākt optimizatora statistikas tieši datu vārdnīcas un galvenajām lietotāju shēmām SYS un SYSTEM. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

DBMS_STATS.GATHER_FIXED_OBJECTS_STATS procedūra tiek lietota tikai pēc datubāzes atjaunināšanas, jauna moduļa implementēšanas, datubāzes parametru mainīšas (piemēram, SGA, PGA) un ja tiek novērota vāja datubāzes veiktspēja. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

DBMS_STATS.GATHER_INDEX_STATS procedūra vāc datubāzes indeksu statistikas. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

DBMS_STATS.GATHER_SCHEMA_STATS procedūra veido objektu statistikas par konkrētu shēmu. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

DBMS_STATS.GATHER_SYSTEM_STATS procedūra vāc datubāzes un operētājsistēmas statistikas, piemēram, par CPU ātrumu. Ir divu veidu iespējamās sistēmas statistikas – vāktas darba slodzes režīmā un atslodzes režīmā. Sistēmas statistikas var vākt tikai vienreiz, ierīkojot sistēmu. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

DBMS_STATS.GATHER_TABLE_STATS procedūra veido tabulas, tās kolonnu un indeksu statistikas. Pieejamo parametru uzskaitījums ir redzams 2.2. tabulā. Parametru pilnu aprakstu skatīt 2.1. tabulā.

Ir svarīgi izvēlēties, kā un cik daudz ir nepieciešams vākt optimizatora statistikas datubāzē. Ja tas tiks darīts pārāk bieži, statistiku vākšanas process var noslogot sistēmu un samazināt datubāzes sniegumu. No otras puses, ja statistikas netiek veidotas vispār vai reti, sistēmas sniegums būs neapmierinošs, laikietilpīgs un resursus aizņemošs.

DBMS_STATS pakotne ir būtiska sastāvdaļa SQL pieprasījumu izpildes snieguma kontrolēšanā. Procedūras un parametri ir viegli lietojami un pārveidojami atbilstoši nepieciešamajiem uzstādījumiem. Ir ieteicams lietot vairāk DBMS_STATS pakotnes procedūras optimizatora statistiku vākšanā un lietot mazāk inicalizēšanas parametru maiņu. [18]

2.2. tabula – **DBMS_STATS pakotnes procedūru un parametru uzskaitījums** [5,6]

	GATHER_DATABASE_STATS	GATHER_DICTIONARY_STATS	GATHER_FIXED_OBJECTS_STATS	GATHER_INDEX_STATS	GATHER_SCHEMA_STATS	GATHER_SYSTEM_STATS	GATHER_TABLE_STATS
ESTIMATE_PERCENT	X	X		X	X		X
BLOCK_SAMPLE	X	X			X		X
METHOD_OPT	X	X			X		X
DEGREE	X	X		X	X		X
CASCADE	X	X			X		X
STATTAB	X	X	X	X	X	X	X
STATID	X	X	X	X	X	X	X
OPTIONS	X	X			X		X
OBJLIST	X	X			X		
STATOWN	X	X	X	X	X	X	X
GATHER_SYS	X						
GRANULARITY	X	X		X	X		X
NO_INVALIDATE	X	X	X	X	X		X
OBJ_FILTER_LIST	X	X			X		
COMP_ID		X					
OWNNAME				X	X		X
INDNAME				X			
PARTNAME				X			X
FORCE				X	X		X
TABNAME							X
STATTYPE							X
GATHERING_MODE						X	
INTERVAL						X	

Datubāzē esošie dati parasti mainās regulāri, tāpēc ir nepieciešams vākt statistikas atbilstoši datu izmaiņu biežumam. [18]

2.1.2. *Microsoft SQL Server 2012* datubāze

Microsoft SQL Server 2012 datubāze piedāvā iespēju vākt optimizatora statistikas gan automātiski, gan manuāli. Šīs statistikas palīdz optimizatoram izveidot augstas kvalitātes pieprasījumu izpildes plānus.

Microsoft piedāvā veidot pilnās tabulas statistikas vai atlasītās statistikas, norādot konkrētus atlases kritērijus. Saturīgi izveidotas atlasītās statistikas salīdzinājumā ar pilnām tabulu statistikām dažos gadījumos var uzlabot pieprasījumu izpildes plāna ātrdarbību. [9]

Optimizators *Microsoft SQL Server* datubāzē vāc statistikas automātiski, ja parametrs AUTO_CREATE_STATISTICS ir ieslēgts. Gadījumos, kad tiek izveidots jauns indeks datubāzē, optimizators arī automātiski savāc statistikas par šo indeksu, tabulu vai skatu. [9]

Atlasītās statistikas var vākt ar komandu CREATE STATISTICS, norādot dažādus parametrus. (skat. 2.3. tabulu) Šo komandu ir ieteicams lietot, ja programma *Database Engine Tuning Advisor* iesaka, ja pieprasījuma atlases kritērija kolonnas nav saistītas ar vienu indeksu, ja SQL pieprasījumā nepieciešamajām tabulām nav vāktas statistikas. [9]

2.3. tabula – **CREATE STATISTICS parametri [9]**

Parametra nosaukums	Apraksts
STATISTICS_NAME	Tiek norādīts statistiku nosaukums.
TABLE_OR_INDEXED_VIEW_NAME	Norāde uz tabulu vai indeksu, kam jāvāc statistikas.
COLUMN[,..n]	Norāde uz tabulas vai indeksa kolonnu vai kolonnām, kam jāvāc statistikas.
WHERE <filter predicate>	Norāda atlases kritēriju tabulas datiem, kam vākt statistikas.
FULLSCAN	Aprēķina statistikas visai tabulai. Šis parametrs ir vienāds ar sample 100 percent parametru.
SAMPLE <number> {PERCENT / ROWS}	Norāda aptuveno procentuālo vērtību vai rindu skaitu tabulā optimizatoram statistiku vākšanas precizēšanai. PERCENT var būt robežās no 0 līdz 100, bet ROWS parametrs var būt robežās no 0 līdz patiesajam rindu skaitam tabulā.
NORECOMPUTE	Izslēdz automātisko statistiku vākšanu

	(AUTO_STATISTICS_UPDATE) konkrētai statistikai. Optimizators savāks statistikas un pēc tam izslēgs tās nākotnes atjaunināšanu.

Ja nesen ir veikta datubāzes atjaunošana, uzlabošana vai pieprasījumu izpildes ātrums ir lēns, ir ieteicams vākt statistikas datubāzē manuāli, izmantojot UPDATE STATISTICS komandu, ja statistikas jau kādreiz ir vāktas objektos, vai izmantojot CREATE STATISTICS, ja statistikas datubāzes objektiem nav veidotas vispār. Tā tiks nodrošināts, ka statistikas nav novecojušas un pieprasījumu izpildes plāns būs korektāks.

2.2. Optimizators

Optimizators ir iebūvēta DBMS programmatūra, kas, izanalizējot dažādus parametrus, nosaka visefektīvāko un labāko veidu, kā izpildīt konkrētu SQL pieprasījumu datubāzē. Optimizators parasti darbojas sistēmas fonā, un lietotājiem nemēdz būt tieša saskare ar to. [5]

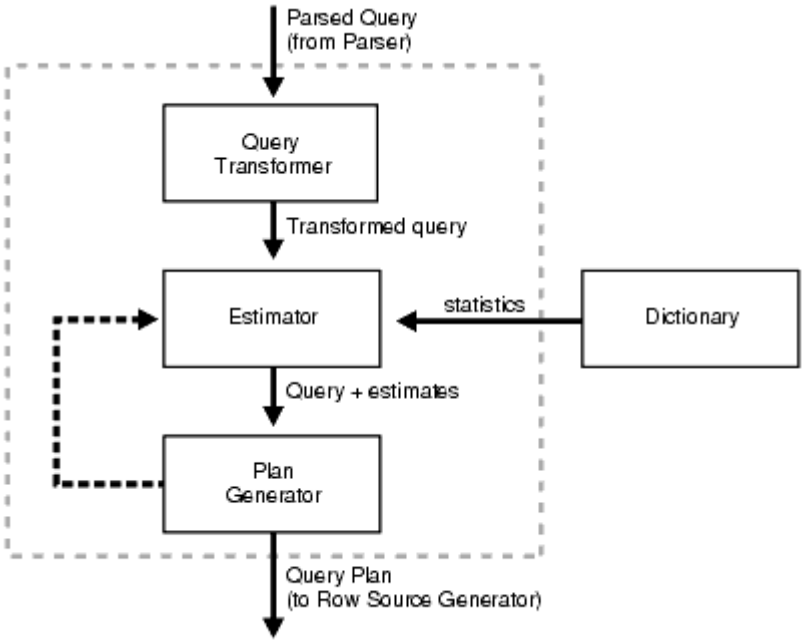
2.2.1. *Oracle 11g* datubāze

Datubāze SQL pieprasījumus var izpildīt dažādos veidos, piemēram, ar pilnu tabulu pārbaudi, priekšmetu indeksa pārbaudi, ligzdstruktūras cikliem, jaucējsavienošanu. SQL pieprasījuma izpildes ātrums ir atkarīgs no optimizatora izvēlētā pieprasījuma izpildes plāna. [5]

Oracle optimizatora darbība pēc lietotāja SQL pieprasījuma iesniegšanas datubāzē visos gadījumos ir vienāda. Pirmajā kārtā iebūvētā programmatūra izveido vairākus iespējamos izpildes plānus, balstoties uz pieejamajiem piekļuves ceļiem. Otrajā kārtā optimizators katram izpildes plānam novērtē vērtību, balstoties uz datubāzes statistikām, kas tiek iegūtas no datubāzes datu vārdnīcas un parāda iepriekš ievāktu informāciju par sistēmu, tās tabulām un rindu skaitu tabulās. Izpildes plāna vērtība ir automātiski aprēķināts lielums, kas atspoguļo SQL pieprasījuma izpildes ātrumu, aprēķinos tiek iekļauti arī tādi sistēmas un operētājsistēmas uzstādījumi kā atmiņas, centrālā procesora, operatīvās atmiņas lielumi u.c. Trešajā kārtā optimizators salīdzina pirmajā kārtā iegūtos izpildes plānus. SQL pieprasījums tiek izpildīts pēc plāna, kam ir viszemākā otrajā kārtā aprēķinātā vērtība. [5]

Optimizators katra SQL pieprasījuma izpildes plāna izvēlē patērē pāris sekundes, tāpēc ir svarīgi iegūt precīzākas tabulu statistikas. Ja tabulas rindu skaits, ko redz sistēma, ir pēc iespējas tuvāks patiesajam tabulas rindu skaitam datubāzē, optimizators ir spējīgs aprēķināt pēc iespējas precīzāku SQL pieprasījuma izpildes plāna ātrumu.

Oracle optimizators sastāv no trīs galvenajām sastāvdaļām – SQL pieprasījumu pārveidotāja, aprēķinātāja un izpildes plāna ģeneratora. (skat. 2.1. att.) [5]



2.1. att. – ***Oracle*** **optimizatora uzbūve** [5]

SQL pieprasījumu pārveidotājs SQL pieprasījumu sadala blokos. Piemēram, iekšējs SELECT pieprasījums tiek uzskatīts par vienu bloku, bet ārējs SELECT pieprasījums tiek uzskatīts par otru bloku. Tā SQL pieprasījumu pārveidotājs redz, ka optimizatoram tiek padoti divi bloki, un lemj, vai šos blokus pārveidot un apvienot, lai pieprasījuma izpilde būtu efektīvāka. SQL pieprasījumu transformators izmanto tādas tehnikas kā datubāzes skatu apvienošanu, predikātu pārveidošanu, apakšpieprasījumu atdalīšanu un pieprasījumu pārrakstīšanu uz materializētajiem skatiem. [5]

Tālāk optimizators pārveidotu SQL pieprasījumu nodod tālāk aprēķinātājam, kas nosaka izpildes plāna vērtību, ņemot vērā selektivitāti, kardinalitāti un vērtību. Selektivitāte parāda filtru no rindām, ja tiek izmantots kāds WHERE pieprasījums. Kardinalitāte parāda kopējo rindu skaitu, vērtība parāda darba vai resursu izmantošanas vienības. [5]

Šajā optimizatora darbības daļā ļoti liela loma ir statistikām. Aprēķinātājs izmanto pēdējās savāktās statistikas, kas atrodas datu vārdnīcā, lai aprēķinātu izpildes plāna selektivitāti, kardinalitāti un vērtību. Jo svaigākas un akurātākas ir statistikas, jo šie aprēķinātāja lielumi ir precīzāki. Aprēķinātāja procesa laikā optimizatora statistikas parāda, cik daudz bloku ir tabulā, cik daudz bloku lasījumu ir nepieciešami SQL pieprasījuma izpildei utt. [8]

Tālāk optimizators pieprasījumu ar aprēķiniem nodod izpildes plāna ģeneratoram, kas izveido dažādus pieprasījuma izpildes plānus, ņemot vērā piekļuves ceļus tabulām, kolonnām, savienošanas metodes un savienošanas secības. [5]

Piekļuves ceļi ir veidi, kā dati tiek izgūti no datubāzes pieprasījuma izpildes plāna laikā. Tabulā 2.4. ir apkopoti *Oracle* piekļuves ceļu veidi. [5]

2.4. tabula – ***Oracle*** **piekļuves ceļu veidi** [5]

Piekļuves ceļu veidi	Apraksts	Kad optimizators izmanto?
Pilna tabulas pārbaude (ang. *Full Table Scan*)	Pārbauda visas rindas tabulā un izņem no saraksta tās rindas, kas neatbilst SQL pieprasījuma atlases kritērijiem. Pilnā tabulas pārbaude ir labāka par indeksu pārbaudi, ja tiek pārbaudītas apjomā lielas tabulas. Pilnā tabulas pārbaude lieto apjomīgākus ievadizvades izsaukumus, un tie ir efektīvāki kā lietojot vairāk mazus ievadizvades izsaukumus.	• Optimizators izmanto pilno tabulas pārbaudi, ja tabulai ir pārāk maz indeksu un SQL pieprasījums nevar izmantot esošos indeksus tabulā. • Optimizators izmanto pilno tabulas pārbaudi, ja tiek uzskatīts, ka pieprasījuma izpildē ir nepieciešams atlasīt lielāko daļu no tabulas. • Optimizators izmanto pilno tabulas pārbaudi, ja tabula ir mazāka par inicializācijas parametra DB_FILE_MULTIBLOCK_READ_COUNT lielumu, kuru datubāze var nolasīt vienā ievadizvades izsaukumā.
Tabulas rindas identifikatora pārbaude (ang. *Table Rowid Scan*)	Katrai tabulas rindai ir identifikators, kas apraksta datu failu un datu bloku, kurā ir konkrētā rinda un kur tā atrodas konkrētajā blokā. Pieprasījuma izpildes laikā meklējot tabulas rindu pēc rindas	Parasti tabulas rindas identifikatora pārbaude seko pēc indeksa rindas identifikatora pārbaudes. Tabulas piekļuve un rindas identifikatora pārbaude var būt nepieciešama dažādām tabulas kolonnām, uz kurām nenorāda indeksi. Ja SQL

		identifikatora, ir ātrākais veids kā atlasīt konkrētu rindu, jo šīs rindas atrašanās vieta ir jau norādīta datubāzē. Vispirms *Oracle* optimizators atrod meklējamo rindu identifikatorus no pieprasījuma atlases kritērijiem vai no indeksu pārbaudes vienam vai vairākiem tabulas indeksiem.	pieprasījuma izpildes laikā indekss satur visas kolonnas un rindas, kas atbilst SQL pieprasījuma atlases kritērijiem, tad tabulas rindas identifikatora pārbaude var tikt neizsaukta.
Indeksu pārbaude (ang. *Index Scan*)	Unikālo indeksu pārbaude (ang. *Index Unique Scan*)	Parasti šis pārbaudes veids atgriež vienu rindas identifikatoru. *Oracle* datubāze izmanto unikālo indeksu pārbaudi, ja SQL pieprasījums satur UNIQUE vai PRIMARY KEY ierobežojumu.	Optimizators izmanto unikālo indeksu pārbaudi, ja SQL pieprasījumā ir definētas kolonnas ar unikāliem indeksiem.
	Indeksu diapazona pārbaude (ang. *Index Range Scan*)	Parasti tiek lietots izlases datu piekļuvei. Atlasītie dati tiek atgriezti augošā secībā pēc indeksiem. Vairākas vienādas atlasītās rindas tiek kārtotas augošā secībā pēc rindas identifikatora.	Optimizators izmanto indeksu diapazona pārbaudi, ja indeksa kolonnas ir norādītas pieprasījuma atlases kritērijā šādā veidā: • col = :b1; • col < :b1; • col > :b1; • iepriekšminētās col vienādības ir savienotas ar AND kombināciju.
	Indeksu diapazona	Parasti tiek lietots izlases datu piekļuvei. Šī pārbaude	Optimizators izmanto indeksu diapazona pārbaudi dilstoši, ja

	pārbaude dilstoši (ang. *Index Range Scan Descending*)	ir identiska indeksu diapozona pārbaudei, tikai atlasītie dati tiek atgriezti dilstošā secībā pēc indeksiem.	indekss apmierina pieprasījuma atlases kritēriju ar norādītu dilstošo parametru.
	Indeksa pilna pārbaude (ang. *Index Full Scan*)	Pilnā indeksu pārbaude pārbauda visus indeksus pēc kārtas. Tā izslēdz kārtošanas operācijas, jo dati tiek sakārtoti pēc indeksa identifikatora.	Optimizators izmanto pilno indeksu pārbaudi, ja: • ORDER BY nosacījumā visas norādītās kolonnas ir indeksā; • GROUP BY nosacījumā visas norādītās kolonnas ir indeksā.
	Indeksa pilna ātrā pārbaude (ang. *Index Fast Full Scan*)	Pilnā indeksa ātrā pārbaude pārbauda visus indeksus sajauktā secībā, kādā tie atrodas uz diska.	Optimizators izmanto pilno indeksu ātro pārbaudi, ja SQL pieprasījums atlasa informāciju tikai no indeksa.

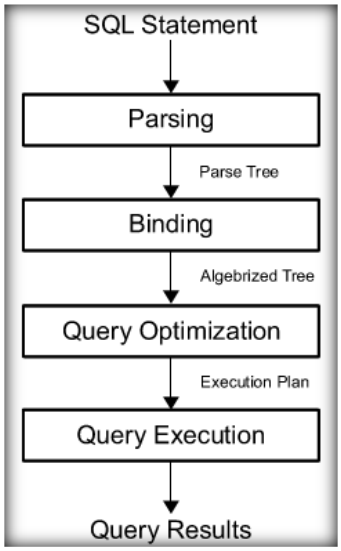
Oracle optimizators izvēlas SQL pieprasījuma izpildes piekļuves ceļu, izvērtējot visus iespējamos variantus. Vispirms optimizators atrod iespējamos piekļuves ceļus, kas apmierina pieprasījuma atlases kritērijus, tad izveido dažādus izpildes plānus un nosaka tiem vērtību. Beigās optimizators izvēlas izpildes plānu ar viszemāko vērtību un ātrāko izpildes laiku. Ir iespējams arī iegūt informāciju par izvēlēto izpildes plānu, izmantojot EXPLAIN PLAN komandu. (skat. 2.3. nodaļā) [5]

Piekļuves ceļa izvēlē ļoti liela nozīme ir optimizatora statistiku atbilstībai esošajai situācijai datubāzē. Ja tabulai statistikas nav vāktas sen, LAST_ANALYZED kolonna tabulā ALL_TABLES būs ar jau sen pagājušu datumu. Ja dati tabulā mainās regulāri, optimizators tabulu uzskatīs par izmērā mazu un izvēlēsies pilno tabulas pārbaudi. Ja tabulā būs krietni vairāk datu nekā to parāda optimizatora statistikas, optimizators izvēlēsies, viņaprāt, ātrāko un efektīvāko pieprasījuma izpildes plānu, bet patiesībā šis plāns var izrādīties ilgākais izpildes ziņā, jo pilnā tabulas pārbaude iet caur visu tabulu un tās rindām, kolonnām. [5]

2.2.1. *Microsoft SQL Server 2012* datubāze

Pieprasījumi tiek padoti *Microsoft SQL Server* datubāzei, izmantojot T-SQL. Galvenais optimizatora uzdevums ir pēc iespējas ātrāk izveidot efektīvus pieprasījuma izpildes plānus. Kā otro galveno uzdevumu var minēt SQL pieprasījuma izpildi datubāzē pēc izvēlētā izpildes plāna soļiem. [10]

Microsoft SQL Server 2012 optimizators sastāv no četrām galvenajām daļām – parsēšanas procesa, saistīšanas procesa, pieprasījuma optimizēšanas un pieprasījuma izpildes. (skat. 2.2. attēlu)



2.2. att. – ***Microsoft SQL Server*** **optimizatora uzbūve** **[10]**

SQL pieprasījums tiek nodots parsēšanas un saistīšanas procesam, kura laikā tiek izveidots loģisks koks. Katra koka mezgls apraksta loģisku operāciju, kuru jāveic pieprasījumam. Piemēram, konkrētas tabulas apskate vai atlases kritēriju ievērošana. Parsēšanas procesa laikā tiek pārbaudīta padotā pieprasījuma sintakse un pareizība. Saistīšanas procesa laikā tiek pārbaudīta nosaukumu atbilstība eksistējošiem objektiem datubāzē. [10]

Tālāk šis pārstrādātais pieprasījuma koks tiek padots pieprasījumu optimizēšanas daļai. Pieprasījumu optimizēšanas procesa laikā tiek atrasti dažādi pieprasījuma izpildes plāna varianti, kuriem ir noteikta vērtība, skatoties uz optimizatora statistikām un citiem datubāzes parametriem. [10]

Microsoft SQl Server 2012 optimizēšanas procesā beigās tiek atrasts optimālākais SQL pieprasījuma izpildes plāns. Pieprasījuma izpilde tiek veikta, izmantojot piekļuves ceļus tabulas datiem un indeksiem. (skat. 2.5. tabulu) *SQL Server* piekļuves ceļi tiek sadalīti divos veidos. Indeksu un tabulu pārbaude iziet cauri visām objekta rindām, bet indeksu un tabulu

sasniegšana izmanto B koku vai fiziskās datu adreses, lai piekļūtu konkrētai rindai indeksā vai tabulā. [11]

2.5. tabula – ***Microsoft SQL Server*** **piekļuves ceļu veidi**[11]

Piekļuves ceļu veidi	Apraksts
Indeksa sasniegšana (ang. *Index Seek*)	Izmanto B koka apstaigāšanu pa mezgliem, lai atrastu nepieciešamo informāciju.
Indeksa pārbaude (ang. *Index Scan*)	Pārskata visas indeksa rindas, kas sakārtotas indeksu secībā. Datubāze lieto, kad SQL pieprasījumā ir norādīts ORDER BY nosacījums.
Mezglota indeksa sasniegšana (ang. *Clustered Index Seek*)	Atrod rindas no mezglota indeksa, izmantojot sasniegšanas mehānismu.
Mezglota indeksa pārbaude (ang. *Clustered Index Scan*)	Pārbauda meglotus indeksus. Atgriež tikai atlasītas rindas, ja ir norādīts WHERE nosacījums. Rindas parāda sakārtotā mezglota indeksa secībā, ja tas ir prasīts. Citādi nav garantēts, ka atgrieztais saraksts būs sakārtots.
Atslēgas uzmeklēšana (ang. *Key Lookup (Clustered)*)	Atgriež vienu rindu no indeksa saraksta. Bieži tiek lietots, ja kolonna ir norādīta ar UNIQUE ierobežojumu.
Atslēgas uzmeklēšana (ang. *RID Lookup (Heap)*)	Atgriež vienu rindu no indeksa saraksta. Rindas meklēšana pēc identifikatora ļauj ātri atrast vēlamo rindu, jo šīs rindas atrašanās vieta jau ir norādīta datubāzē.
Pilna tabulas pārbaude (ang. Full *Table Scan*)	Pārskata visas tabulas rindas un kolonnas.

Kā pēdējais solis tiek ņemta pieprasījuma izpilde pēc izpildes plāna, kas apraksta darbības un algoritmus, pēc kuriem ir iespējams iegūt nepieciešamo informāciju no datubāzes. [10]

2.3. Pieprasījuma izpildes plāns

Datubāze SQL pieprasījuma izpildes laikā veic dažādus soļus, kas tiek apkopoti pieprasījuma izpildes plānā, parādot pieprasījumu izpildes gaitu un piekļuves ceļus.

2.3.1. *Oracle 11g* datubāze

Oracle datubāzes pieprasījuma izpildes plānu var apskatīt ar EXPLAIN PLAN priekšrakstu. Izpildot šo komandu, rezultātā tiek iegūta tabula, kurā ērti un vienkārši ir aprakstīts SQL pieprasījuma izpildes plāns ar piekļuves ceļiem. [5]

Automātiski datubāzes uzstādīšanas laikā tiek izveidots PLAN TABLE, kas satur EXPLAIN PLAN priekšraksta izvades informāciju visiem lietotājiem katram pieprasījuma izpildes plānam. PLAN TABLE tiek izveidots kā publisks sinonīms katram lietotājam uz globālu datubāzes pagaidu tabulu. Šo tabulu var izveidot arī manuāli ar *catplan.sql* instrukciju virkni. [5]

Oracle iesaka izdzēst un pārbūvēt lokālo PLAN TABLE, ja ir notikusi datubāzes atjaunināšana uz jaunāku versiju, jo var gadīties, ka starp datubāžu versijām, tabulas kolonnas un funkcionalitāte var būt nedaudz pārveidota. [5]

Lai iegūtu pieprasījuma izpildes plānu, nepieciešams tikai SQL pieprasījumam pievienot EXPLAIN PLAN FOR priekšrakstu. (skat. 2.3. attēlu)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT *
  3  FROM   emp e, dept d
  4  WHERE  e.deptno = d.deptno
  5  AND    e.ename  = 'SMITH';

Explained.

SQL>
```

2.3. att. – **Piemērs pieprasījuma izpildes plāna iegūšanas komandai konkrētam SQL pieprasījumam** [8]

Kad izpildes plāns ir paskaidrots, informāciju ir nepieciešams iegūt no PLAN TABLE. To var izdarīt trīs dažādos veidos:

1. izmantojot *utlxpls.sql* instrukciju virkni (skat. 2.4. attēlu);
2. izmantojot *utlxplp.sql* instrukciju virkni, ko var palaist ar komandu *@$ORACLE_HOME/rdbms/admin/utlxplp.sql*;

3. izmantojot DBMS_XPLAN.DISPLAY tabulas funkciju, ko var palaist ar komandu SELECT * FROM TABLE (DBMS_XPLAN.DISPLAY( )).

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxpls.sql

Plan Table
--------------------------------------------------------------------------------
| Operation                 |  Name    |  Rows | Bytes|  Cost  | Pstart| Pstop |
--------------------------------------------------------------------------------
| SELECT STATEMENT          |          |       |      |        |       |       |
|  NESTED LOOPS             |          |       |      |        |       |       |
|   TABLE ACCESS FULL       |EMP       |       |      |        |       |       |
|   TABLE ACCESS BY INDEX RO|DEPT      |       |      |        |       |       |
|    INDEX UNIQUE SCAN      |PK_DEPT   |       |      |        |       |       |
--------------------------------------------------------------------------------

8 rows selected.

SQL>
```

2.4. att. – **Piemērs izpildes plāna iegūšanai ar utlxpls.sql skritpu** [8]

Attēlā 2.4. ir attēlots SQL pieprasījuma izpildes plāns. Kolonnā OPERATIONS tiek norādīts, ka pieprasījuma izpildē tiks izmantota pilnā tabulu pārbaude EMP tabulai un rindas identifikatora pārbaude DEPT tabulai. Informācija tiks atlasīta, izmantojot arī unikālo indeksu pārbaudi PK_DEPT indeksam. EXPLAIN PLAN komanda neizpilda SQL pieprasījumu datubāzē. Tā tikai parāda pieprasījuma izpildes gaitu, piekļuves ceļus, savienošanas metodes un savienošanas secības informatīvos nolūkos.

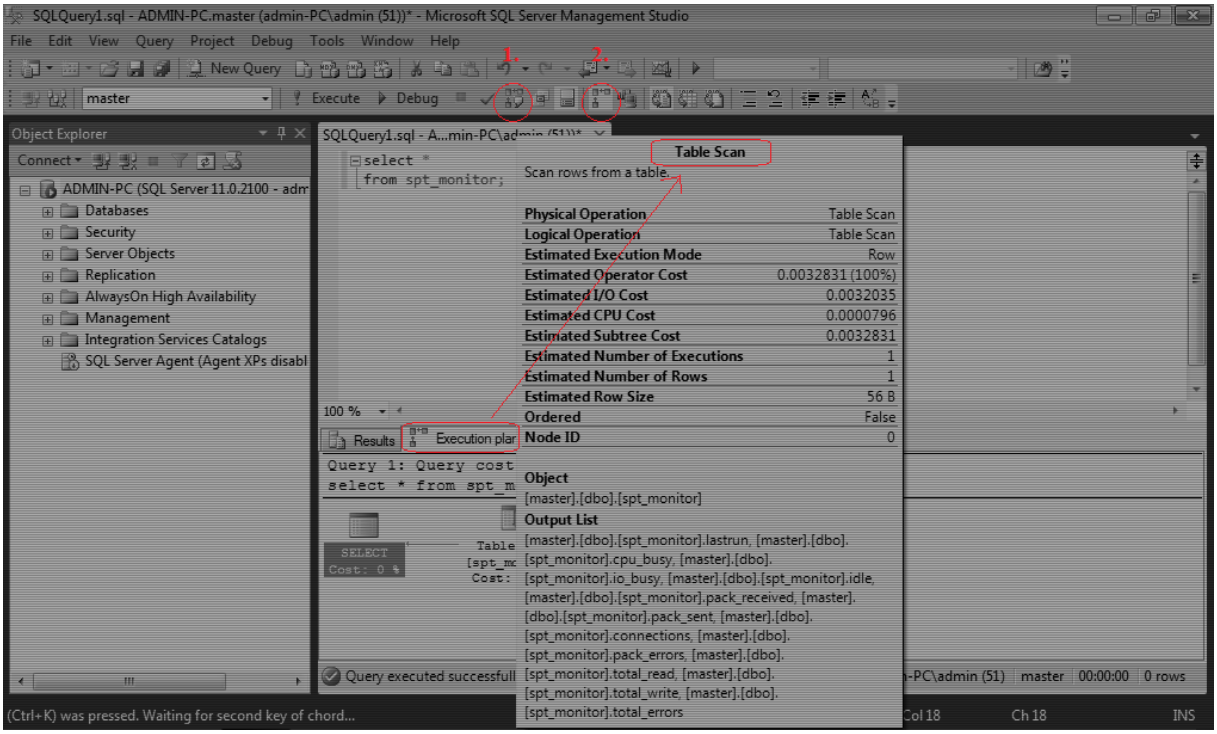
2.3.2. *Microsoft SQL Server 2012* datubāze

Ir divi veidi, kā *Microsoft SQL Server 2012* attēlo pieprasījuma izpildes plānus.

Grafisko veidu var izmantot ar *Management Studio*. Izpildes plāna apskats ir ērts un intuitīvs. Tomēr šim apskates veidam ir arī mīnuss – izpildes plānā izmantotā piekļuves ceļa plašākas informācijas apraksts ir pieejams tikai kursoru novietojot uz attiecīgā piekļuves ceļa ikonas. Lai grafiski attēlotu izpildes plānu *Management Studio* rīkjoslā ir jānospiež pirmā poga, kas rezultātu sadaļā izvada izpildes plāna grafisku zīmējumu un papildus informāciju. Nospiežot otro pogu, izpildes plāns parādīsies nākamajā reizē, kad tiks palaists attiecīgais SQL pieprasījums. (skat. 2.5. att.) [11]

Pēc SQL pieprasījuma izpildes cilnē *Results* ir iespējams apskatīt SQL pieprasījuma iznākumu, bet cilne *Execution plan* parāda izpildes plānu grafiski. Ir iespējams redzēt piekļuves ceļu nosaukumus ar atbilstošām tabulām un indeksiem, kas tiek izmantoti

konkrētajā solī. Gadījumos, kad SQL pieprasījuma uzbūve ir sarežģītāka, ir iespējams redzēt izpildes plānu, kas parāda piekļuves ceļu secības un saistību savā starpā. (skat. 2.5. att.) [11]



2.5. att. – Management Studio izpildes plāna grafiska attēlošana [11]

Otrs veids ir iegūt pieprasījuma izpildes plānu tabulāri, izmantojot *Management Studio*. Lai to izdarītu, ir jāizpilda SET STATISTICS PROFILE ON komanda. Tabulārais izpildes plāns uzreiz pēc SQL pieprasījuma izpildes ir pieejams rezultātu sadaļā kā papildus rindas ar kolonnām ROWS, EXECUTES, STMTTEXT, STMTID un NODEID. Šis apskates veids ir neparocīgs, jo kolonnas STMTTEXT saturs ir grūti salasāms. Bet, ja ir nepieciešamība, izpildes plāna koda tekstu dublēt, tabulārais apskates veids ir ērtāks par grafisko apskates veidu. [11]

2.4. Optimizatora parametri

Optimizatora darbību ir iespējams kontrolēt ar dažādiem parametriem, kas nosaka tā darbību dažādās situācijās.

2.4.1. *Oracle 11g* datubāze

Oracle datubāzes pamatā ir inicializēšanas parametri, kas atrodas inicializācijas failā un tiek palaisti pie datubāzes sāknēšanas. Šie parametri nosaka, kā un kādos gadījumos strādā datubāze un tās programmas.

Oracle dod iespēju kontrolēt optimizatora darbību ar pieciem inicializēšanas parametriem. (skat. 2.6. tabulu) [5]

2.6. tabula – ***Oracle*** **inicializēšanas parametri optimizatora kontrolei** [5]

Inicializēšanas parametrs	Apraksts
CURSOR_SHARING	Pārveido literātiskas vērtības SQL pieprasījumos par saistošiem mainīgajiem. Šī pārveidošana uzlabo kursoru koplietošanu un var ietekmēt SQL pieprasījumu izpildes plānus.
DB_FILE_MULTIBLOCK_READ_COUNT	Norāda bloku skaitu, kas tiek nolasīti vienā ievadizvades operācijā, kamēr notiek pilnā tabulas pārbaude vai ātrā pilnā indeksu pārbaude.
OPTIMIZER_INDEX_COST_ADJ	Regulē indeksu vērtību izpildes plāna izvēles laikā. Iespējamās vērtības ir no 1 līdz 10000. Noklusētā vērtība ir 100, kas nozīmē, ka indeksi tiek vērtēti pēc vispārpieņemta normāla vērtības modeļa un piekļuves ceļiem. Ja vērtība ir 10, tad indeksa piekļuves ceļa vērtība ir uzskatāma par desmit reizes zemāku nekā ir vispārpieņemts.
OPTIMIZER_MODE	Uzstāda optimizatora režīmu, kad tiek sāknēta datubāze. Iespējamās vērtības ir ALL_ROWS, FIRST_ROWS_n un FIRST_ROWS.
PGA_AGGREGATE_TARGET	Kontrolē datubāzes atmiņas lielumu, kas tiek piešķirts kārtošanai un jausējsavienojumiem. Jo lielāks ir piešķirtās atmiņas lielums, jo optimizators izpilda pieprasījumu ātrāk.

Ar inicializēšanas parametru mainīšanu ir jābūt ļoti uzmanīgam, jo tie ir galvenie *Oracle* datubāzi raksturojošie parametri. Ja kāds tiek mainīts neapdomīgi, var tikt ietekmēti arī citi paramteri, kas nodrošina datubāzes korektu darbību.

2.4.2. *Microsoft SQL Server 2012* datubāze

Microsoft SQL Server 2012 datubāze piedāvā optimizatora darbības kontrolēšanai trīs parametrus, kas nosaka, kad un kā tiek vāktas pilnās tabulu statistikas. Šos parametrus var mainīt tikai datubāzes līmenī. [9]

Ja AUTO_CREATE_STATISTICS parametrs ir ieslēgts, optimizators automātiski vāc kolonnu un tabulu statistikas, ja tas ir nepieciešams pieprasījuma izpildes plānam. Šis parametrs nevāc atlasītās statistikas un statistikas indeksiem. Tas konkrēti strādā ar kolonnu statistikām tabulās. [9]

Ja AUTO_UPDATE_STATISTICS parametrs ir ieslēgts, optimizators automātiski noskaidro, kad tabulu statistikas ir novecojušas, un atjauno tās nakamajā reizē, kad statistikas ir nepieciešamas pieprasījuma izpildē. Parasti statistikas kļūst novecojušas pēc INSERT, UPDATE, DELETE, MERGE operācijām, kad tiek mainīti dati tabulās. Šis parametrs vāc atlasītās statistikas, statistikas indeksiem un izveidotās statistikas ar CREATE STATISTICS komandu. [9]

AUTO_UPDATE_STATISTICS_ASYNC ir asinhrons statistiku vākšanas parametrs, kas parāda, vai optimizators izmanto sinhronas vai asinhronas statistikas. Noklusētā vērtība šim parametram ir sinhrona, kas nozīmē, ka pieprasījumu izpildes plāns tiek izvēlēts ar aktuālām statistikām. Ja statistikas ir novecojušas, tad optimizators gaida, kad statistikas tiks atjauninātas, un tikai pēc tam izpilda SQL pieprasījumu. Ja ir norādīts asinhrons režīms, tad SQL pieprasījums tiek izpildīts arī ņemot vērā novecojušas statistikas. [9]

2.5. Teorijas salīdzinājums

Teorijas daļā tika apskatītas *Oracle 11g* un *Microsoft SQL Server 2012* datubāžu optimizatora statistikas un procedūras, optimizatora darbības principi un uzbūve, SQL pieprasījuma izpildes plāns un optimizatoru regulējošie datubāzes parametri.

Oracle 11g datubāze statistiku vākšanai piedāvā DBMS_STATS pakotni ar septiņām iespējamām procedūrām un kopā 23 maināmiem parametriem. (skat. 2.1. tabulu) *Microsoft SQL Server 2012* piedāvā automātisku statistiku vākšanu, ieslēdzot parametru AUTO_CREATE_STATISTICS, piedāvā arī manuāli ar komandām CREATE STATISTICS un UPDATE STATISTICS un 7 komandu parametriem (skat. 2.3. tabulu).

Oracle optimizators strādā idejiski vienādi kā *Microsoft SQL Server* optimizators. Vienīgā atšķirība ir procesa soļu nodalīšanā. Salīdzinot 2.1. un 2.2. attēlus redzams, ka *Oracle* optimizatora sastāvdaļas ir vienkāršāk sadalītas nekā *Microsoft SQL Server*. *Oracle* SQL pieprasījumu pārveidotājs atbilst *Microsoft* parsēšanas un saistīšanas procesiem. Aprēķinātāja un izpildes plāna ģeneratora soļi atbilst pieprasījuma optimizēšanas daļai. *Microsoft* vēl atsevišķi nodala pieprasījuma izpildes procesu.

Pieprasījuma izpildes plāna ģenerēšana atšķiras abās datubāzēs. *Oracle 11g* piedāvā EXPLAIN PLAN komandu, pēc kuras izpildes ir vēl nepieciešams apskatīt PLAN TABLE tabulu, lai redzētu konkrētā SQL pieprasījuma izpildes plānu. Bet *Microsoft SQL Server 2012* sniedz gan grafiskas, gan tabulāras apskates iespējas. Neraugoties uz to, ka tabulārā iespēja ir grūti salasāma un neparocīga, grafiskā pieprasījuma izpildes plāna apskates iespēja sniedz plašu informāciju par konkrēto SQL pieprasījumu un tā izpildi.

Oracle 11g piedāvā piecus inicializācijas parametrus (skat. 2.6. tabulu), kas palīdz kontrolēt optimizatora darbību datubāzes līmenī. Ar šo parametru maiņu ir jābūt piesardzīgiem, jo, mainot kādu parametru, ir iespējams mainīt datubāzes darbību citā līmenī. Piemēram, piešķirot PGA_AGGREGATE_TARGET lielāku atmiņas vietu, tiek samazināts SGA lielums, kas nodrošina pašas datubāzes darbību. Šāds solis var novest pie ļoti lēnas datubāzes darbības un pieprasījumu apstrādes. *Microsoft SQL Server 2012* piedāvā trīs parametrus optimizatora darbības kontrolei. (skat. 2.4.2. nodaļu) Šie parametri nevar kritiski ietekmēt datubāzes darbību, jo tie ir paredzēti tieši statistiku vākšanas un optimizatora darbības kontrolēšanai.

Pēc teorijas apskates salīdzinājuma var secināt, ka *Oracle 11g* datubāze piedāvā plašākas optimizācijas iespējas un komandas nekā *Microsoft SQL Server 2012*.

3. PRAKTISKĀ DAĻA

Pamatojoties uz vispārējo *Oracle* un *Microsoft SQL Server* salīdzinājumu, praktiskajā daļā tiks izmantotas sistēmu versijas, kas ir vecākas par 2014. gadu. Tās ir *Oracle 11g* un *Microsoft SQL Server 2012*.

Praktiskajā daļā ir plānots apskatīt *Oracle* datubāzes HR shēmu, kas tiek pārnesta arī uz *Microsoft SQL Server 2012* datubāzi, optimizatora statistiku vākšanas komandas pirms un pēc datu ievades tabulās un konkrēta SQL pieprasījuma izpildes plāna uzbūvi pirms un pēc statistiku vākšanas un datu ievades tabulās.

Ir izveidots arī praktiskās daļas plāns, pēc kura vadoties, tiek veikts darbs. (skat. 3.1. tabulu)

3.1. tabula – **Praktiskās daļas plāns**

Npk.	Soļa nosaukums	Soļa apraksts
1.	Vides sagatavošana	Uz *VirtualBox* diviem diskiem tiek uzstādīta *Windows* 7 32-bit operētājsistēma.
2.	Datubāžu uzstādīšana	Uz viena *VirtualBox* diska tiek uzstādīta *Oracle 11g* datubāze, uz otra – *Microsoft SQL Server 2012* datubāze.
3.	HR shēmas izveide	*Oracle* datubāzē tiek izveidota HR shēma, iegūts relāciju modelis un DDL izveides skripts. Izmanotjot *Data Modeler* rīku, HR shēmas izveides skripts no SQL valodas tiek pārveidots atbilstoši *Microsoft SQL Server 2012* shēmas izveides instrukciju virknei.
4.	Statistiku vākšana	Tiek savāktas pirmās statistikas abās datubāzēs ar idejiski vienādām komandām, atzīmēts izpildes ātrums un kvalitāte.
5.	Datu ievade tabulās	Abās datubāzēs tiek ievadīti sākotnējie dati ar 100000 rindām.
6.	Izpildes plāns SQL pieprasījumam	Tiek izveidots sarežģīts SQL pieprasījums HR shēmā. Izpildes plāni tiek salīdzināti abās datubāzēs.
7.	Statistiku vākšana	Statistiku vākšana pēc pirmās datu ievades tabulās.
8.	Izpildes plāns SQL pieprasījumam	Tiek izpildīts tas pats SQL pieprasījums HR shēmā. Izpildes plāni tiek salīdzināti abās datubāzēs.

9.	Datu ievade tabulās	Abās datubāzēs tiek ievadīti dati ar 100000 rindām.
10.	Izpildes plāns SQL pieprasījumam	Tiek izpildīts tas pats SQL pieprasījums HR shēmā. Izpildes plāni tiek salīdzināti abās datubāzēs.

Pēc praktiskās daļas pirmo desmit soļu izpildes tiek atkārtoti 7. līdz 10. solis trīs reizes. Šāda shēma palīdzēs praktiski parādīt SQL pieprasījumu izpildes plāna ātrumu pirms un pēc statistiku vākšanas un datu ievades tabulās. Kā arī parādīs statistiku korektumu pirms un pēc apjomīgas datu ievades tabulās.

3.1. Datubāžu uzstādīšana

Datubāzes tiks instalētas katra uz sava *VirtualBox* diska. Kā tika noskaidrots DBMS vispārējā salīdzinājuma nodaļā, vienīgais kopīgais servera operētājsistēmu atbalsts ir ar *Windows* 7 32-bit versiju, ko tika nolemts izmantot labākas SQL datubāžu salīdzinājuma pārskatāmības dēļ.

3.1.1. *Oracle 11g* datubāzes uzstādīšana

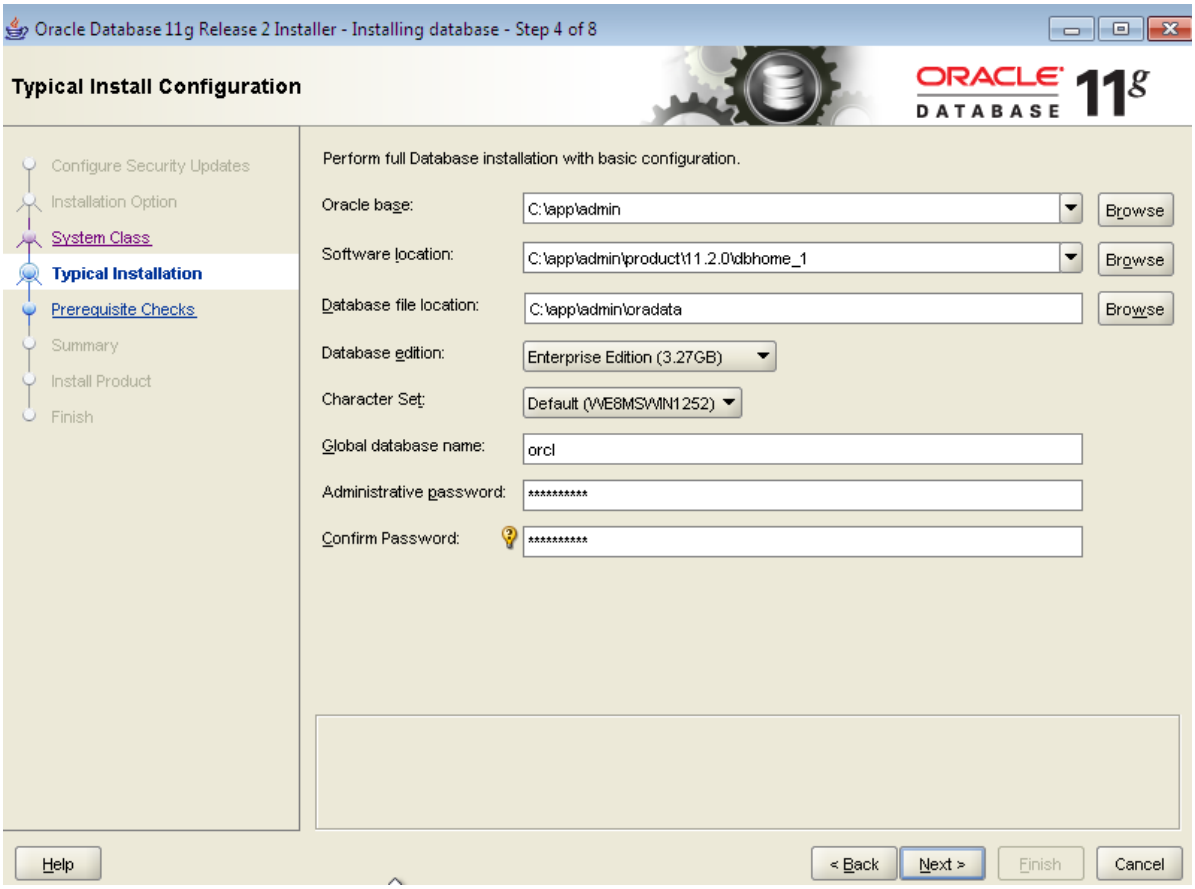
Pirms *Oracle 11g Enterprise* datubāzes instalācijas uz *Windows 7 32-bit* versijas operētājsistēmas ir nepieciešams izpildīt uzstādīšanas prasības, lai datubāze varētu darboties atbilstoši pamata uzstādījumiem.

Uzstādīšanas prasības: [7]

- Aparatūra:
  - Procesors – *Intel (x86), AMD64, Intel EM64T*
  - RAM – 2GB
  - Virtuālā atmiņa – 2GB
  - Atmiņas lielums diskā – vismaz 5.35GB
  - Ekrāna izšķirtspēja – 1024 x 768
  - Pagaidu direktorijas lielums – 500MB
  - *Oracle* mājas direktorijas lielums – 3.6GB
  - Datu direktorija lielums – 1.9GB
- Programmatūra:
  - Operētājsistēmu atbalsts – izvēlēts *Windows 7 Professional*
  - Tīkla protokols – TCP/IP
- Strādājošs saimniekdatora nosaukums;

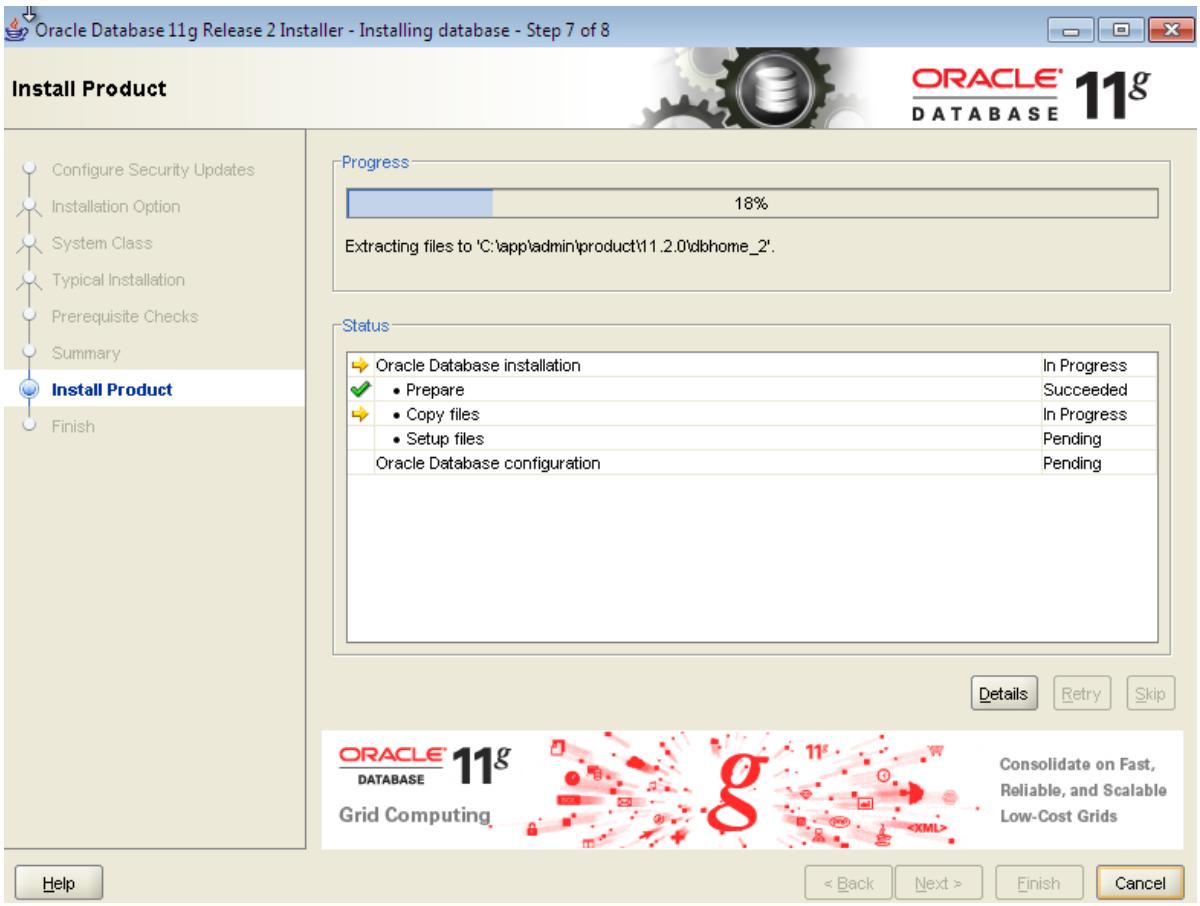
Gadījumā, ja nav izpildīta kāda no pamata uzstādīšanas prasībām, *Oracle* automātiski datubāzes uzstādīšanas laikā pārbauda šīs prasības un dod paziņojumu, ja kāda nav izpildīta.

Oracle 11g datubāzes uzstādīšana sastāv no divām datnēm, kas tiek novietotas vienā mapē un atarhivētas. Lai palaistu OUI, nepieciešams tikai atvērt *setup.exe* datni. Datubāzes uzstādīšanas process ir vienkāršs, ērts un intuitīvs. Kā galveno uzstādīšanas soli var minēt konfigurācijas daļu, kurā ir jānorāda galveno mapju nosaukumus, datubāzes nosaukumu un administratīvo paroli. (skat. 3.1. att.) [7]



3.1. att. – ***Oracle 11g* datubāzes uzstādīšanas konfogurācijas daļa** [7]

Pēc konfigurācijas daļas izpildes OUI pārbauda uzstādīšanas prasību izpildi. Ja visas prasības ir ievērotas, OUI sāk *Oracle 11g* datubāzes uzstādīšanu. Gadījumā, ja kāda no prasībām nav izpildīta, datubāzes uzstādīšanas process neturpina darbu. (skat. 3.2. att.)



3.2. att. – **Oracle 11g datubāzes uzstādīšana** [7]

OUI procesa laikā sagatavo operētājsistēmas uzstādījumus, kopē datnes no atarhivētās mapes uz sistēmu, sāknē iestatnes instrukciju virknes un konfigurē *Oracle* datubāzes pamata un tīkla uzstādījumus.

3.1.2. *Microsoft SQL Server 2012* datubāzes uzstādīšana

Pirms *Microsoft SQL Server 2012 Enterprise* datubāzes uzstādīšanas uz *Windows 7 32-bit* versijas operētājsistēmas ir nepieciešams izpildīt uzstādīšanas prasības, lai datubāze varētu darboties atbilstoši pamata uzstādījumiem.

Uzstādīšanas prasības: [12]

- Aparatūra:
  - Procesors – *Pentium III-compatible* vai ātrāks
  - RAM – 1GB
  - Atmiņas lielums diskā – vismaz 6GB
  - Ekrāna izšķirtspēja – *Super-VGA* (800x600) vai augstākas izšķirtspējas ekrāns
- Programmatūra:

- Operētājsistēmu atbalsts – izvēlēts *Windows 7 Professional*
- Tīkla protokols – TCP/IP
- *.NET Framework*

Microsoft SQL Server 2012 uzstādīšana sastāv no vienas atarhivētas mapes. Lai palaistu uzstādīšanas procesu, nepieciešams atvērt *setup.exe* datni. Pēc sistēmas prasību izpildes pārbaudes tiek atvērts *SQL Server Installation Center* logs, kas ļauj apskatīt datubāzes plānošanas, uzstādīšanas, uzturēšanas, programmatūru, resursu un papildināto uzstādījumu sadaļas. [13]

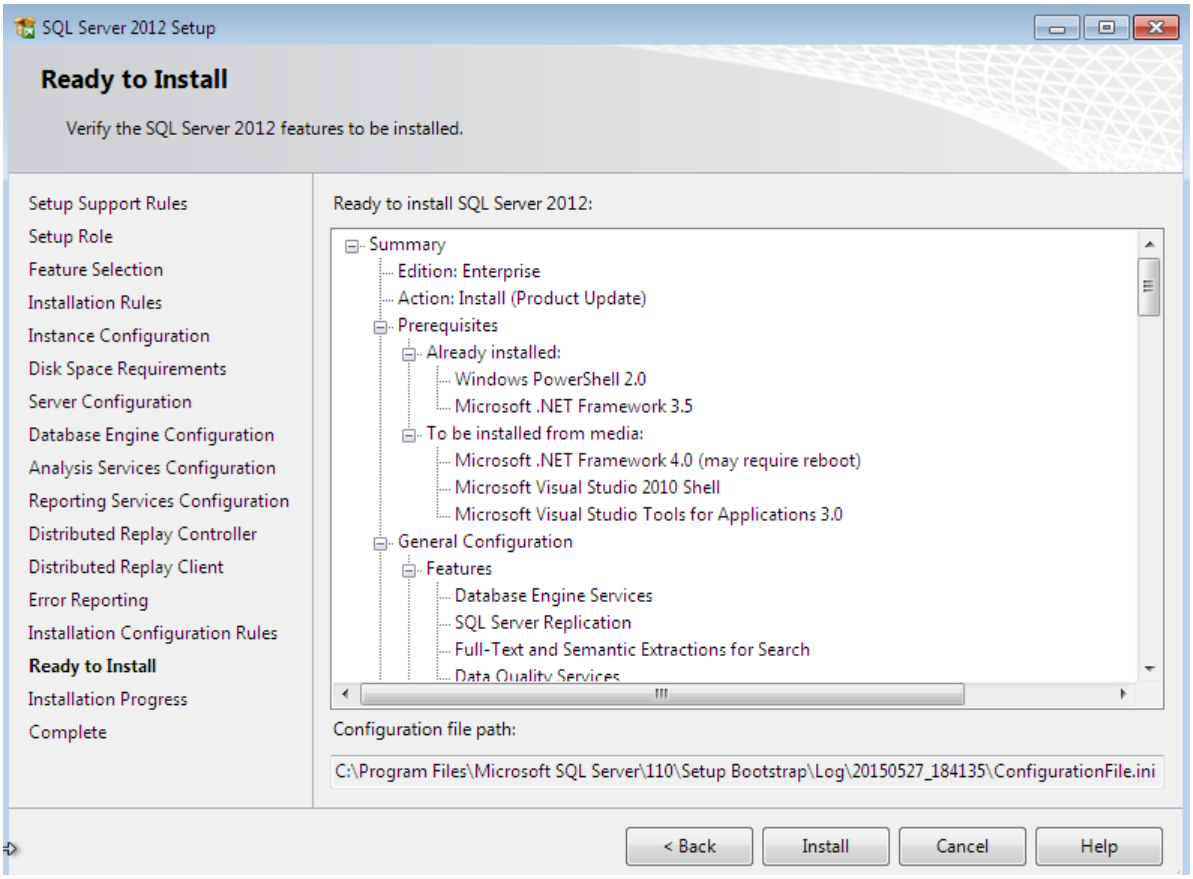
Tiek atvērta uzstādīšanas sadaļa un izvēlēta „*New SQL Server stand-alone installation or add features to an existing installation*" iespēja. (skat. 3.3. att.)

3.3. att. – ***SQL Server Installation Center*** **programmatūra** [13]

Sekojošos soļos tiek apskatītas tādas uzstadīšanas sadaļas kā „*Setup Support Rules*", „*Product Key*", „*License Terms*", „*Product Updates*" un „*Install Setup Files*". Kā nākamais solis tiek veikta pogas „*Install*" izvēle, kas parāda vēl izpildāmos soļus pirms *Microsoft SQL Server 2012* datubāzes uzstādīšanas. [13]

Tālāk jāatzīmē pāris uzstādījumi un paroles tādos soļos kā „*Setup Support Rules*", „*Setup Role*", „*Feature Selection*", „*Instance Configuration*", „*Server Configuration*". Pēc šo

soļu izpildes tiek izdots paziņojums, ka *SQL Server Installation Center* programmatūra ir gatava uzstādīt *Microsoft SQL Server 2012* datubāzi. (skat. 3.4. att.) [13]



3.4. att. – ***SQL Server Installation Center* uzstādīšanas gatavības apstiprinājums** [13]

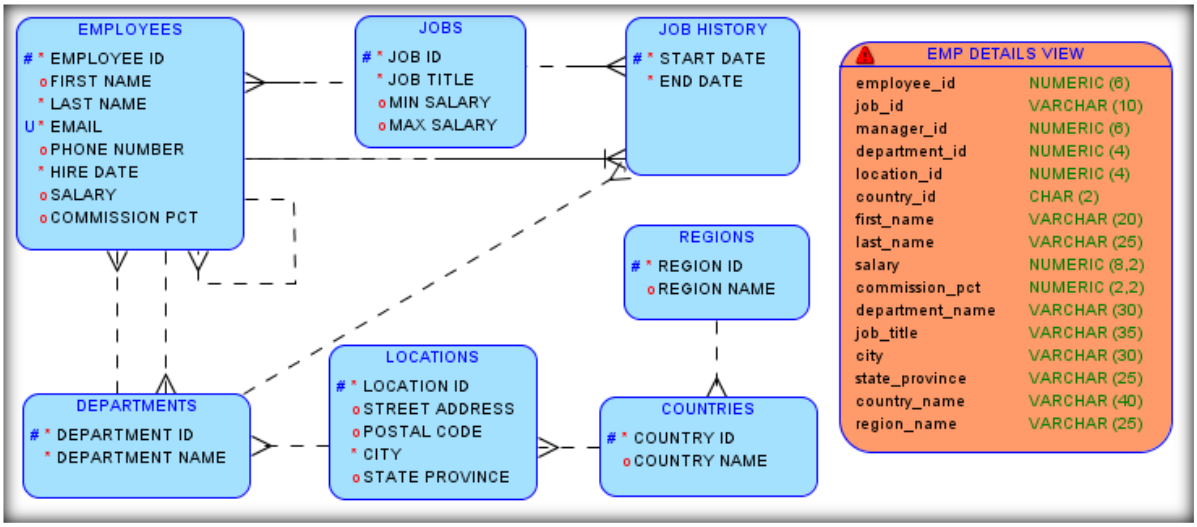
Microsoft SQL Server 2012 uzstādīšana beidzas ar apstiprinājuma logu.

3.2. HR shēma

Oracle piedāvā iespēju datubāzē uzstādīt arī parauga shēmu ar tabulām, indeksiem, secībām, skatiem. Praktiskās daļas ietvaros ir izvēlēta HR shēma, kas apraksta kāda uzņēmuma cilvēkresursu daļas informāciju par darbiniekiem un pieejamajiem amatiem uzņēmumā. [14]

HR shēmā tiek glabāta informācija par darbinieku, viņa vārds un uzvārds, e-pasta adrese, identifikācijas numurs, alga, darba vadītājs utt. Ir iespējams arī atrast ziņas par pieejamajiem amatiem uzņēmumā, minimālajām un maksimālajām algām konkrētā amatā. Ir arī apskatīti dažādādi gadījumi, piemēram, darbinieks uzņēmumā strādā jau ilgi un amatus ir nomainījis vairākkārt, katrs darbinieks strādā konkrētā uzņēmuma nodaļā ar unikālu nosaukumu vai identifikācijas numuru, nodaļas atrodas konrētā ģeogrāfiskā vietā, kuras HR shēmā tiek arī paskaidrotas ar pilnām adresēm.

HR shēmu uzskatāmāk var attēlot ar loģisko relāciju modeli. (skat. 3.5. att.) Pilnu relāciju modeli var aplūkot 1. pielikumā.



3.5. att. – **Loģiskais relāciju modelis HR shēmai** [14]

HR shēma sastāv no: [14]

- viena skata;
- divām procedūrām;
- diviem trigeriem;
- trīs secībām;
- septiņām tabulām;
- deviņpadsmit indeksiem.

EMP_DETAILS_VIEW ir HR shēmas skats, kas parāda uzreiz visu nepieciešamo informāciju, kas būtu jāzina par katru darbinieku. Vieglāk ir atlasīt datus no skata, nevis rakstīt vairākus SQL pieprasījumus katrai tabulai, lai noskaidrotu vēlamo informāciju. Šis skats norāda darbinieka vārdu, uzvārdu, darbinieka, amata, vadītāja, nodaļas, vietas, valsts identifikācijas numuru datubāzē, darbinieka algu, nodaļas, amata nosaukumu, pilsētu, reģionu, valsti, kurā darbinieks dzīvo un strādā. (skat. 3.5. att.) [14]

ADD_JOB_HISTORY ir procedūra, kas pievieno ierakstu JOB_HISTORY tabulā par darbinieku, kas ir nomainījis amatu darba vietā. SECURE_DML ir procedūra, kas kontrolē, vai darbinieks, veicot izmaiņas datubāzē, dara to pēc darba laika standartiem. Ja tas notiek virsstundu laikā, šī procedūra parāda kļūdas paziņojumu, ka veikt izmaiņas ir iespējams tikai darba devēja noteiktās darba laika stundās un nedēļas dienās. [14]

SECURE_EMPLOYEES un UPDATE_JOB_HISTORY ir trigeri, kas kontrolē ADD_JOB_HISTORY un SECURE_DML procedūru izpildes sākumu. [14]

DEPARTMENTS_SEQ, EMPLOYEES_SEQ, LOCATIONS_SEQ ir datubāzē izveidotas secības, kas atvieglo unikālu ierakstu ievadi tabulās. [14]

HR shēmā ir septiņas tabulas. [14] COUNTRIES tabula sniedz informāciju par valsti, kurā atrodas uzņēmuma nodaļa. Kolonna COUNTRY_ID ir tabulas primārā atslēga, kas datubāzē tiek definēta kā unikāls COUNTRY_C_ID_PKX indekss. Citu indeksu tabulai COUNTRIES nav.

REGIONS tabula parāda uzņēmuma nodaļas atrašanās valsts reģionu. Kolonna REGION_ID ir tabulas primārā atslēga, kurai ir izveidots viens unikāls REG_ID_PKX indekss.

LOCATIONS tabula apraksta uzņēmuma nodaļas atrašanās vietu, adresi, pilsētu. Ir pieejami 4 indeksi, kas norāda uz 4 tabulas kolonnām. LOCATIONS primārā atslēga ir unikāls LOC_ID_PKX indekss, pārējie var būt neunikāli. LOC_STATE_PROV_IX un LOC_CITY_IX norāda uz STATE_PROVINCE un CITY kolonnām, bet LOC_COUNTRY_IX ir LOCATIONS tabulas COUNTRY_ID sekundārās atslēgas indekss.

DEPARTMENTS tabula sniedz informāciju par katru uzņēmuma nodaļu, tās nosaukumu, atrašanās vietu un nodaļas vadītāju. DEPT_ID_PKX ir tabulas primārās atslēgas indekss, bet DEPT_LOCATION_IX ir sekundārās atslēgas indekss uz LOCATIONS tabulu.

JOB_HISTORY tabula uzskaita darbiniekus, kas ir mainījuši amatu uzņēmuma ietvaros. JHIST_ID_DATE_PKX ir saliktas primārās atslēgas indekss, kas apvieno EMPLOYEE_ID un START_DATE kolonnas. Tabulai JOB_HISTORY ir definēti vēl trīs indeksi (JHIST_DEPT_IX, JHIST_EMP_IX, JHIST_JOB_IX), kas norāda uz sekundārajām atslēgām.

JOBS tabula parāda visus iespējamos amatus uzņēmumā un tā nodaļās, minimālās un maksimālās algas katram amatam. Pieejams ir tikai primārās atslēgas indekss ar nosaukumu JOB_ID_PKX.

EMPLOYEES tabula sniedz informāciju par katru darbinieku, tā adresi, kontaktinformāciju, algu. Šī tabula uzbūves ziņā ir vislielākā no visām tabulām HR shēmā. Tai ir seši indeksi, no kuriem primārās atslēgas indeksam EMP_EMP_ID_PKX un unikālo ierakstu kolonnas indeksam EMP_EMAIL_UK ir jābūt unikāliem. EMP_NAME_IX, EMP_ MANAGER_IX, EMP_JOB_IX un EMP_DEPARTMENT_IX ir tabulas EMPLOYEES sekundāro atslēgu indeksi.

Ar plašāku HR shēmas tabulu aprakstu, kolonnu nosaukumiem un datu tipiem var iepazīties 4. pielikumā. [14]

3.3. Datu ievade, statistiku vākšana un izpildes plānu ģenerēšana *Oracle* un *Microsoft SQL Server 2012* datubāzē

Sākuma situācijā datubāzēs ir uzstādīta tukša HR shēma ar septiņām tabulām. Pilns tabulu izveides skripts *Oracle* datubāzei ir pieejams 2. pielikumā un *Microsoft SQL Server* datubāzei ir pieejams 3. pielikumā. Vispirms tiek savāktas sākuma optimizatora statistikas, jo nesen uzstādītā datubāzes sistēmā nav informācija par tabulām, indeksiem vai citiem objektiem. Tā kā praktiskajā daļā tiek apskatīta tikai HR shēma, optimizatora statistikas tiek vāktas tikai šim objektam, izmantojot DBMS_STATS pakotnes procedūru ar parametriem, kas idejiskā līmenī ir pielīdzināmi arī *Microsoft SQL Server 2012* datubāzes statistiku vākšanas komandas CREATE STATISTICS parametriem. Attēlā 3.6. ir redzama optimizatora statistiku vākšanas komanda *Oracle 11g* datubāzē, izmantojot komandrindu.

```
SQL>
SQL> execute dbms_stats.gather_schema_stats (ownname => 'HR', estimate_percent =
> 100, options => 'GATHER');

PL/SQL procedure successfully completed.

Elapsed: 00:00:09.49
SQL>
```

3.6. att. – **Statistiku vākšana *Oracle* datubāzē, izmantojot DBMS_STATS pakotni**

GATHER_SCHEMA_STATS procedūras OWNNAME parametrs norāda shēmu, kurai ir jāvāc statistikas, ESTIMATE_PERCENT norāda procentuālo rindu novērtējumu pēc kura vadīties. Ir izvēlēts 100 procentīgs rindu novērtējums, jo shēmas lielums ir mazs, tabulās nav ierakstu un komanda izpildīsies salīdzinoši īsā laika posmā. OPTIONS parametrs GATHER norāda, ka statistikas tiks vāktas pilnīgi visiem objektiem norādītajā shēmā. Ar detalizētāku parametru aprakstu var iepazīties 2.1.1. nodaļā. Šī komanda izpildījās 9.49 milisekundēs, jo HR shēma ir izmērā neliela, tajā ir tikai septiņas tabulas bez datiem.

Microsoft SQL Server 2012 datubāzē pirmreizējās statistikas tiek vāktas ar CREATE STATISTICS komandu, norādot statistikas, tabulas nosaukumu un kolonnas, bet tālākie statistiku vākšanas soļi tiek veikti ar UPDATE STATISTICS komandu, norādot tikai tabulas nosaukumu. (skat. 3.7. attēlu)

```
CREATE STATISTICS stat1
ON dbo.table_name (column_name, column_name)
WITH SAMPLE 100 PERCENT;

UPDATE STATISTICS dbo.table_name;
```

3.7. att. – **Statistiku vākšana *Microsoft* datubāzē**

Optimizatora statistikas ar 3.6. un 3.7. attēlos redzamajām komandām tiek vāktas starp datu ievadēm tabulās, lai būtu iespējams novērot statistiku vākšanas ātruma izmaiņas atkarībā no datu apjoma tabulās.

HR shēmas tabulās REGIONS, COUNTRIES, LOCATIONS, DEPARTMENTS, EMPLOYEES, JOBS, JOB_HISTORY tiek ievadīti sākuma dati ar INSERT INTO TABLE VALUES komandu. Kopumā praktiskās daļas laikā tabulā EMPLOYEES tiek ievadīti 400107 rindu, REGIONS – 4, COUNTRIES – 25, LOCATIONS – 23, DEPARTMENTS – 27, JOBS – 19, JOB_HISTORY – 10. Tālākos soļos dati tiek ievadīti tikai EMPLOYEES tabulā, jo nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Ar konkrētām datu ievades komandām var iepazīties 5. pielikumā.

Oracle optimizatora statistiku vākšanas kvalitāti var uzzināt, izmantojot divus lielumus. Tiek skatīta DBA_TABLES tabulas NUM_ROWS kolonna, kas parāda tabulas rindu skaitu no sistēmas skatu punkta, salīdzinājumā ar patieso tabulas rindu skaitu, izmantojot COUNT(*) komandu. [16] Turpretī *Microsoft* salīdzina SHOW_STATISTICS parametra ROWS vērtību ar patieso tabulas rindu skaitu, izmantojot COUNT(*). Optimizatora statistiku procentuālā novirze tabulām tiek aprēķināta pēc 3.1. formulas:

$$\frac{Patiesie\ \ dati - sistēmas\ \ dati}{Patiesie\ \ dati} * 100\% \quad (3.1.)$$

Procentuālās novirzes HR shēmas tabulām tiek apkopotas pirms un pēc statistiku vākšanas datubāzē, lai būtu iespējams precīzāk noteikt, cik kvalitatīvi ir savāktas optimizatora statistikas datubāzē. Tā kā dati tika ievadīti tikai EMPLOYEES tabulā, procentuālās novirzes mainījās tikai šai tabulai. Attēlā 3.8. ir attēlotas procentuālās novirzes aprēķināšanā izmantotās komandas tabulai REGIONS. [16]

```
select count(*) from HR.REGIONS;
select num_rows from dba_tables where table_name ='REGIONS' and owner = 'HR';
```

3.8. att. – **Tabulas REGIONS procentuālās novirzes aprēķināšanā izmantotās komandas** [16, 18]

HR shēmā uzreiz pēc datu ievades tabulā un optimizatora statistiku atjaunošanas tiek noskaidroti trīs SQL pieprasījumu izpildes plāni. Šādi būs iespējams novērtēt, vai izpildes plāna struktūra mainās, mainot datu apjomu datubāzē un papildinot rindu skaitu tabulās no sistēmas skatu punkta. 6. pielikumā ir parādīti trīs izmantotie SQL pieprasījumi. [17]

Pirmais SQL pieprasījums parāda nodaļas nosaukumu un darbinieku skaitu katrā no tām. Otrais pieprasījums apkopo informāciju par amatu, nodaļu, darbinieku un darba sākuma

datumu katram amatam no 1999. gada līdz 2000. gadam. Pēdējais SQL pieprasījums parāda amatu un vidējo algu, ko saņem darbinieki. [17]

Ar *Oracle 11g* datubāzes praktiskās daļas pilniem darbību soļiem var iepazīties 7. Pielikumā, bet 8. pielikumā ir attēloti *Microsoft SQL Server 2012* datubāzes praktiskās daļas soļi. Piecu praktisko soļu izpildes laikā optimizatora statistikas tika vāktas sešas reizes, kopā tabulā EMPLOYEES tika ievadītas 400107, procentuālās novirzes tika rēķinātas 10 reizes un katram SQL pieprasījumam tika izveidoti 10 izpildes plāni. Ar rezultātu salīdzinājumu var iepazīties 3.5. nodaļā.

3.4. Praktiskās daļas rezultātu salīdzinājums

Šajā nodaļā ir apkopots praktiskās daļas rezultātu salīdzinājums un secinājumi gan datubāzes robežās, gan starp *Oracle 11g* un *Microsoft SQL Server 2012* datubāzēm.

3.4.1. *Oracle 11g* datubāzes rezultāti

Oracle datubāzes pētnieciskais process tika veikts atbilstoši iepriekš izstrādātam testēšanas plānam. Tabulā 3.2. ir apkopoti visi iegūtie rezultāti.

3.2. tabula – ***Oracle*** **praktiskās daļas rezultātu kopsavilkums**

Darbības nosaukums	**1. soļa rezultāti**	**2. soļa rezultāti**	**3. soļa rezultāti**	**4. soļa rezultāti**	**5. soļa rezultāti**
Pirmreizējā statistiku vākšana	9.42 ms	-	-	-	-
Procentuālās novirzes	100%	99.89%	49.99%	33.32%	69.96%
Pirmā SQL pieprasījuma izpildes plāna izmaiņas pēc datu ievades tabulā	-	-	-	-	-
Otrā SQL pieprasījuma izpildes plāna izmaiņas pēc datu ievades tabulā	-	-	-	-	-
Trešā SQL pieprasījuma izpildes plāna izmaiņas pēc datu ievades tabulā	-	-	-	-	-
Statistiku vākšana	2.79 ms	7.67 ms	56.84 ms	22.74 ms	42.90 ms
Procentuālās novirzes	0%	0%	0%	0%	0%
Pirmā SQL pieprasījuma izpildes plāna izmaiņas pēc statistiku	-	Ir izmaiņas	-	-	-

atjaunošanas					
Otrā SQL pieprasījuma izpildes plāna izmaiņas pēc statistiku atjaunošanas	-	Ir izmaiņas	-	-	-
Trešā SQL pieprasījuma izpildes plāna izmaiņas pēc statistiku atjaunošanas	Ir izmaiņas	-	-	-	-

Apkopojot praktiskajā daļā iegūto informāciju, ir iespējams izvirzīt secinājumus, kas atspoguļo optimizatora uzvedību, darbības ātrumu un kvalitāti dažādās situācijās.

Pēc datiem no 3.3. tabulas var teikt, ka optimizatora statistiku vākšanas ātrums lielā mērā ir atkarīgs no izvēlētajiem parametriem un datu apjoma tabulās. Statistiku pirmreizējā vākšana uzreiz pēc datubāzes uzstādīšanas aizņem ilgāku laiku nekā pirmā statistiku vākšana pēc datu ievades tabulās. Ir iespējams novērot statistiku vākšanas ātruma samazinājumu pēc katru 100 000 rindu ievades datubāzes tabulā, jo ar katru datu ievadi analizējamo datu apjoms pieaug.

Statistiku vākšana datubāzē uzlabo izpildes plāna kvalitāti, jo optimizators izpildes plāna izvēles procesa laikā redz aktuālāku datubāzes tabulu un rindu daudzumu. Tādā veidā optimizators var izvēlēties ātrāko izpildes plānu, jo tam ir pieejama atjaunotāka informācija. To apliecina arī procentuālās novirzes aprēķināšanas solis. Pēc datu ievades tabulā sistēma redz krietni mazāku tabulas apjomu nekā patiesībā. Bet uzreiz pēc optimizatora statistiku atjaunošanas procentuālā novirze kļūst mazāka.

Pēc praktiskās daļas veikšanas tika novērots, ka katrs SQL pieprasījums mainīja izpildes plānu vienu reizi. Pirmā un otrā SQL pieprasījuma izpildes plāns mainījās 2. soļa laikā, bet trešā SQL pieprasījuma izpildes plāns jau mainījās 1. solī. Optimizators visiem SQL pieprasījumiem mainīja izpildes kārtību tieši pēc statistiku atjaunošanas datubāzē. Jau pēc pirmās datu ievades tabulā optimizators izlēma trešo SQL pieprasījumu izpildīt ar plašākiem indeksu piekļuves ceļiem. No sākuma tas veica *full table access* un *index range scan.* Bet, pieaugot datu apjomam tabulā, nav prātīgi veikt *full table access*, tapēc izpildes plāns tika pārveidots, ņemot *hash join, index fast full scan, index unique scan* piekļuves ceļus. Var secināt, ka *Oracle* optimizators SQL pieprasījuma izpildes plāna izvēlē balstās vairāk uz statistikām, tādējādi izvēloties ātrākos un efektīvākos izpildes plānus.

3.4.2. *Microsoft SQL Server 2012* datubāzes rezultāti

Microsoft datubāzes praktiskā daļa tika veikta atbilstoši iepriekš izstrādātam testēšanas plānam. Tabulā 3.3. ir apkopoti visi iegūtie rezultāti.

3.3. tabula – ***Microsoft*** **praktiskās daļas rezultātu kopsavilkums**

Darbības nosaukums	**1. soļa rezultāti**	**2. soļa rezultāti**	**3. soļa rezultāti**	**4. soļa rezultāti**	**5. soļa rezultāti**
Pirmreizējā statistiku vākšana	10.02 ms	-	-	-	-
Procentuālās novirzes	100%	99.89%	98.75%	98.75%	75.43%
Pirmā SQL pieprasījuma izpildes plāna izmaiņas pēc datu ievades tabulā	-	Ir izmaiņas	-	-	-
Otrā SQL pieprasījuma izpildes plāna izmaiņas pēc datu ievades tabulā	-	-	-	-	-
Trešā SQL pieprasījuma izpildes plāna izmaiņas pēc datu ievades tabulā	-	Ir izmaiņas	-	-	-
Statistiku vākšana	4.56 ms	6.54 ms	7.03 ms	7.04 ms	12.17 ms
Procentuālās novirzes	0%	0%	0%	0%	0%
Pirmā SQL pieprasījuma izpildes plāna izmaiņas pēc statistiku atjaunošanas	-	-	-	-	-
Otrā SQL pieprasījuma izpildes plāna izmaiņas pēc statistiku atjaunošanas	-	-	-	-	-
Trešā SQL pieprasījuma izpildes plāna izmaiņas pēc statistiku atjaunošanas	-	-	-	-	-

Apkopojot informāciju, kas tika savākta praktiskajā daļā, ir iespējams izvirzīt secinājumus, kas atspoguļo *Microsoft SQL Server 2012* datubāzes optimizatora uzvedību, darbības ātrumu un kvalitāti dažādās situācijās.

Statistiku vākšanas ātrumam samazinoties datu apjoma pieauguma dēļ, procentuālās novirzes arī samazinās. Jo ilgāk tiek vāktas optimizatora statistikas datubāzē, jo kvalitatīvākas tās tiek glabātas datubāzē.

Arī *Microsoft SQL Server 2012* datubāzē optimizatora statistiku vākšanas ātrums ir atkarīgs no datu apjoma tabulās. Pie lielākas fiziskās atmiņas izmantošanas datubāzē, apstrādājamie dati un tabulas pieaug skaitliskā ziņā. Ir nepieciešams ilgāks laiks, lai apstrādātu visus datus.

Pēc informācijas apkopošanas tika noskaidrots, ka optimizators ir mainījis SQL izpildes plānus pirmajam un trešajam pieprasījumam 2. soļa laikā tieši pēc datu ievades tabulā. SQL izpildes plānus var apskatīt 8. pielikumā.

REZULTĀTI

Darbā tika izpētītas un salīdzinātas *Oracle 11g* un *Microsoft SQL Server 2012* datubāžu optimizācijas iespējas un parametri. Ir plaši pieejama informācija par pašām datubāzēm, to uzstādījumiem, komandām, bet ir apgrūtināti atrast vienuviet apkopotu informāciju par optimizatoru un tā darbību. *Oracle* datubāzes avoti sniedz plašāku ieskatu risināmajā problēmā nekā par *Microsoft* pieejamajos materiālos.

Darbā risināma problēma ir izpētīta un ir gūts priekšstats par populārāko datubāžu optimizācijas iespējām, optimizatora darbības principiem un uzbūvi. Šo darbu var uztvert kā informatīvu materiālu, kas apraksta datuzbāžu optimizācijas procesu un tā salīdzinājumu starp dažādām datubāzēm.

Optimizators ir iebūvēta datubāžu pārvaldības sistēmas programmatūra gan *Oracle* datubāzē, gan *Microsoft SQL Server 2012* datubāzē, kas, izanalizējot dažādus parametrus, nosaka visefektīvāko un labāko veidu, kā izpildīt konkrētu SQL pieprasījumu datubāzē. Lielākajā daļā gadījumu cilvēki, kas strādā ar konkrētu datubāzi, neiedziļinās optimizācijas procesā un tā iespējās tikai līdz brīdim, kad datubāzes darbība un veiktspēja ir salīdzinoši mazāka nekā iepriekš. Ir jāmāk atšķirt, kad datubāzes veiktspēja ir zemāka dēļ optimizatora darbības. Vairumā gadījumu vainīgi ir servera uzstādījumi un uzbūve, neprecīza tīkla konfigurācija vai noslodzes lielums.

Statistiku vākšana datubāzē uzlabo izpildes plāna kvalitāti, jo optimizators izpildes plāna izvēles procesa laikā redz aktuālāku datubāzes tabulu un rindu daudzumu. Tādā veidā optimizators var izvēlēties ātrāko izpildes plānu, jo tam ir pieejama atjaunotāka informācija.

Ir novērojamas SQL pieprasījumu izpildes plānu maiņas pirms statistiku vākšanas salīdzinājumā ar optimizatora statistiku atjaunošanu. Optimizators ir spējīgs izvēlēties efektīvāko izpildes plānu, pamatojoties uz informācijas daudzumu datubāzē par tabulu, rindu un indeksu apjomu.

Pēc praktiskās daļas salīdzinājuma ir redzams, ka *Oracle* optimizācijas process ir ātrāks un kvalitatīvāks, optimizators apdomīgāk izvēlas SQL pieprasījumu izpildes plānus nekā *Microsoft* optimizators.

Pēc teorijas apskates salīdzinājuma var secināt, ka *Oracle 11g* datubāze piedāvā plašākas optimizācijas iespējas, komandas un parametrus nekā *Microsoft SQL Server 2012*. Jo plašāka ir pieejamā informācija, jo vieglāk ir izvēlēties statistikas vākšanas veidu un apjomu konkrētā datubāzē.

SECINĀJUMI

Ir ieteicams ieguldīt līdzekļus un spēku optimizācijas procesu izpētē ar mērķi uzlabot datubāzes darbību. Kvalitatīvas optimizatora statistikas dod iespēju datubāzei strādāt ātrāk un efektīvāk. Ir svarīgi izvēlēties, kā un cik daudz ir nepieciešams vākt optimizatora statistikas datubāzē. Ja tas tiks darīts pārāk bieži, statistiku vākšanas process var noslogot sistēmu un samazināt datubāzes sniegumu. No otras puses, ja statistikas netiek veidotas vispār vai reti, sistēmas sniegums būs neapmierinošs, laikietilpīgs un resursus aizņemošs.

Šī darba praktiskā daļa sastāvēja tikai no pieciem soļiem ar aptuveni 400 000 rindu ievadi tabulā, kas ļāva idejiski novērot optimizatora darbību un SQL pieprasījumu izpildes plānu izvēli pirms un pēc statistiku atjaunošanas datubāzē, pirms un pēc lielas datu ievades. Lai būtu iespējams veikt kvalitatīvus secinājumus un argumentāciju, balstītus uz praktisko daļu, būtu nepieciešama datu ievade visās HR shēmas tabulās lielākā apjomā. Latvijas Universitātes Datorikas fakultātes kursa darba ietvaros tika pētīts *Oracle* optimizators uzņēmuma produkcijas vidē, kura sastāvēja pat no 14 000 000 rindu vienā tabulā. Pēc tik lielas praktiskās daļas pārbaudes, optimizators uzrādīja procentuālo noviržu starpības un dažādus lielumus arī pēc statistiku vākšanas. Bija iespēja plašāk izanalizēt iegūtos rezultātus, parametru dažādību un optimizatora darbību salīdzinājumā ar reālu datubāzes serveri un tā noslodzi. Tālāk pētot optimizācijas iespēju salīdzinājumu dažādās datubāzēs būtu lietderīgi apkopot informāciju vēl par pāris populārākajām datubāzēm, dažādām to versijām un to darbību uz reāliem serveriem un darbā ar cilvēkiem.

Šis darbs kalpo kā informatīvs materiāls visiem interesentiem, kam ir nepieciešama optimizatora darbības regulēšana gan *Oracle 11g,* gan *Microsoft SQL Server 2012* datubāzē.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Solid IT. (2015.g.). "Method of calculating the scores of the DB-ENGINES ranking" [Tiešsaiste]. Pieejams: http://db-engines.com/en/ranking_definition (Atsauce: 05.05.2015).

[2] Solid IT. (2015.g.). "DB-ENGINES Ranking" [Tiešsaiste]. Pieejams: http://db-engines.com/en/ranking (Atsauce: 05.05.2015).

[3] Solid IT. (2015.g.). "DB-ENGINES Ranking – Trend Popularity" [Tiešsaiste]. Pieejams: http://db-engines.com/en/ranking_trend (Atsauce: 05.05.2015).

[4] Solid IT. (2015.g.). "System Properties Comparison Microsoft SQL Server vs. Oracle" [Tiešsaiste]. Pieejams: http://db-engines.com/en/system/Microsoft+SQL+Server%3BOracle (Atsauce: 05.05.2015).

[5] *Performance Tuning Guide, 11g Release 2 (11.2),* Oracle Corp., 2011

[6] Mahmmoud Adel. (2012.g., Novembris). "Database Administation Tips. All About Statistics In Oracle" [Tiešsaiste]. Pieejams: http://dba-tips.blogspot.com/2012/11/all-about-statistics-in-oracle.html (Atsauce: 07.05.2015)

[7] *Installation Guide, 11g Release 2 (11.2) for Microsoft Windows,* Oracle Corp., 2014

[8] *Expert Oracle SQL. Optimization, Deployment and Statistics*, Tony Hasler, 2014

[9] Microsoft. (2015.g.). "Statistics" [Tiešsaiste]. Pieejams: https://msdn.microsoft.com/en-us/library/ms190397%28v=sql.110%29.aspx (Atsauce: 10.05.2015)

[10] Benjamin Nevarez. (2012.g., Augusts). "The SQL Server Query Optimizer" [Tiešsaiste]. Pieejams: https://www.simple-talk.com/sql/sql-training/the-sql-server-query-optimizer/ (Atsauce: 10.05.2015)

[11] Markus Winand. (2010.g., Jūnijs). “Use The Index, Luke” [Tiešsaiste]. Pieejams: http://use-the-index-luke.com/sql/explain-plan (Atsauce: 11.05.2015)

[12] Microsoft. (2015.g.). “Hardware and Software Requirements for Installing SQL Server 2012” [Tiešsaiste]. Pieejams: https://msdn.microsoft.com/en-us/library/ms143506%28v=sql.110%29.aspx (Atsauce: 12.05.2015)

[13] Sivakumar Vellingiri. (2012.g., Marts). “SQL Server: Step by step installation guide for SQL Server 2012 ” [Tiešsaiste]. Pieejams: http://sivasqlbi.blogspot.com/2012/03/sql-server-step-by-step-installation.html (Atsauce: 14.05.2015)

[14] *Sample Schemas, 11g Release 2 (11.2),* Oracle Corp., 2010

[15] *Technical Comparison of Oracle Database 11g and SQL Server 2008: Focus on Manageability,* Oracle White Paper, 2008

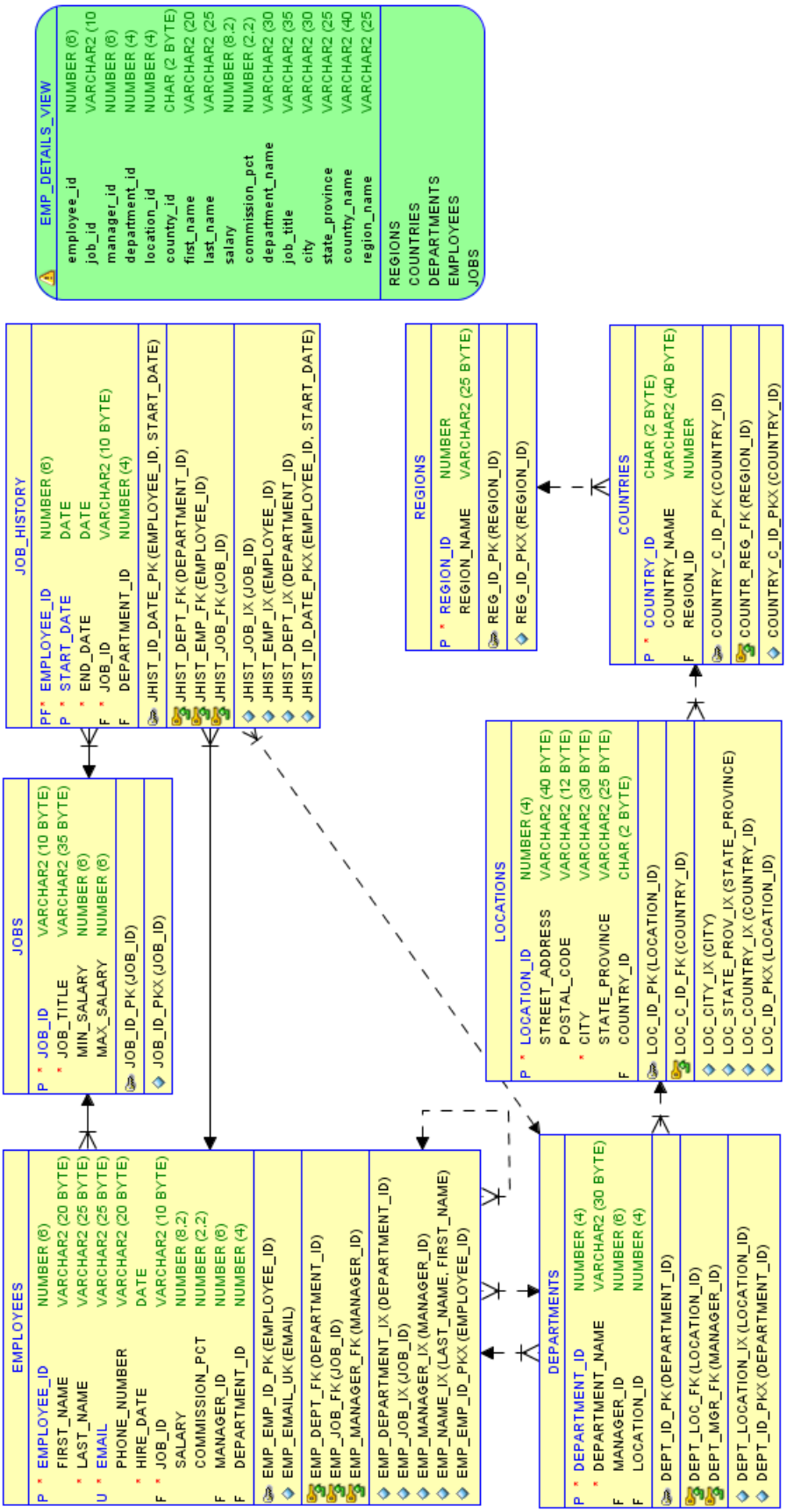
[16] QueryCSM. (2013.g., Marts). “How reliable it is using DBMS_STATS. AUTO_SAMPLE_SIZE for gathering table statistics? ” [Tiešsaiste]. Pieejams: https://querycsm.wordpress.com/2013/03/27/how-reliable-it-is-using-dbms_stats-auto_sample_size-for-gathering-table-statistics/ (Atsauce: 20.05.2015)

[17] Srikanth Pragada. (2011.g., Decembris). “Exercises Related To HR Schema in Oracle Database 11g” [Tiešsaiste]. Pieejams: http://www.srikanthtechnologies.com/oracle/dec9/hrqueries.html (Atsauce: 21.05.2015)

[18] Lauma Šivare, „Statistiku vākšanas ātruma un rezultāta uzlabošana Oracle datubāzē 11gR2”, kursa darbs, 2014.g.

PIELIKUMI

1. pielikums – Relāciju modelis HR shēmai



2. pielikums – Tabulu izveide *Oracle* datubāzē

```
CREATE TABLE JOB_HISTORY(
   EMPLOYEE_ID   NUMBER (6) NOT NULL ,
   START_DATE    DATE NOT NULL ,
   END_DATE      DATE NOT NULL ,
   JOB_ID        VARCHAR2 (10 BYTE) NOT NULL ,
   DEPARTMENT_ID NUMBER (4)) LOGGING;
ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_DATE_CHECK CHECK
(end_date > start_date);

CREATE INDEX JHIST_JOB_IX ON JOB_HISTORY(JOB_ID ASC)LOGGING;
CREATE INDEX JHIST_EMP_IX ON JOB_HISTORY(EMPLOYEE_ID ASC)LOGGING;
CREATE INDEX JHIST_DEPT_IX ON JOB_HISTORY(DEPARTMENT_ID
ASC)LOGGING;
CREATE UNIQUE INDEX JHIST_ID_DATE_PKX ON JOB_HISTORY(EMPLOYEE_ID
ASC, START_DATE ASC);
ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_ID_DATE_PK PRIMARY
KEY(EMPLOYEE_ID, START_DATE);

CREATE TABLE COUNTRIES(
   COUNTRY_ID   CHAR (2 BYTE) NOT NULL ,
   COUNTRY_NAME VARCHAR2 (40 BYTE) ,
   REGION_ID    NUMBER)LOGGING;

CREATE UNIQUE INDEX COUNTRY_C_ID_PKX ON COUNTRIES(COUNTRY_ID
ASC);
ALTER TABLE COUNTRIES ADD CONSTRAINT COUNTRY_C_ID_PK PRIMARY
KEY (COUNTRY_ID);

CREATE TABLE DEPARTMENTS(
   DEPARTMENT_ID   NUMBER (4) NOT NULL ,
   DEPARTMENT_NAME VARCHAR2 (30 BYTE) NOT NULL ,
   MANAGER_ID      NUMBER (6) ,
   LOCATION_ID     NUMBER (4))LOGGING;

CREATE INDEX DEPT_LOCATION_IX ON DEPARTMENTS(LOCATION_ID
ASC)LOGGING;
CREATE UNIQUE INDEX DEPT_ID_PKX ON DEPARTMENTS(DEPARTMENT_ID
ASC);
ALTER TABLE DEPARTMENTS ADD CONSTRAINT DEPT_ID_PK PRIMARY KEY
(DEPARTMENT_ID);

CREATE TABLE EMPLOYEES(
   EMPLOYEE_ID    NUMBER (6) NOT NULL ,
   FIRST_NAME     VARCHAR2 (20 BYTE) ,
   LAST_NAME      VARCHAR2 (25 BYTE) NOT NULL ,
   EMAIL          VARCHAR2 (25 BYTE) NOT NULL ,
   PHONE_NUMBER   VARCHAR2 (20 BYTE) ,
   HIRE_DATE      DATE NOT NULL ,
   JOB_ID         VARCHAR2 (10 BYTE) NOT NULL ,
   SALARY         NUMBER (8,2) ,
```

```
   COMMISSION_PCT NUMBER (2,2) ,
   MANAGER_ID     NUMBER (6) ,
   DEPARTMENT_ID  NUMBER (4))LOGGING;
ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_SALARY_MIN CHECK
(salary>0);

CREATE INDEX EMP_DEPARTMENT_IX ON EMPLOYEES(DEPARTMENT_ID
ASC)LOGGING;
CREATE INDEX EMP_JOB_IX ON EMPLOYEES(JOB_ID ASC)LOGGING;
CREATE INDEX EMP_MANAGER_IX ON EMPLOYEES(MANAGER_ID
ASC)LOGGING;
CREATE INDEX EMP_NAME_IX ON EMPLOYEES(LAST_NAME ASC, FIRST_NAME
ASC)LOGGING;
CREATE UNIQUE INDEX EMP_EMP_ID_PKX ON EMPLOYEES(EMPLOYEE_ID
ASC);
ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_EMP_ID_PK PRIMARY KEY
(EMPLOYEE_ID);
ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_EMAIL_UK UNIQUE
(EMAIL);

CREATE TABLE JOBS(
   JOB_ID     VARCHAR2 (10 BYTE) NOT NULL ,
   JOB_TITLE  VARCHAR2 (35 BYTE) NOT NULL ,
   MIN_SALARY NUMBER (6) ,
   MAX_SALARY NUMBER (6))LOGGING;

CREATE UNIQUE INDEX JOB_ID_PKX ON JOBS(JOB_ID ASC);
ALTER TABLE JOBS ADD CONSTRAINT JOB_ID_PK PRIMARY KEY (JOB_ID);

CREATE TABLE LOCATIONS(
   LOCATION_ID    NUMBER (4) NOT NULL ,
   STREET_ADDRESS VARCHAR2 (40 BYTE) ,
   POSTAL_CODE    VARCHAR2 (12 BYTE) ,
   CITY           VARCHAR2 (30 BYTE) NOT NULL ,
   STATE_PROVINCE VARCHAR2 (25 BYTE) ,
   COUNTRY_ID     CHAR (2 BYTE))LOGGING;

CREATE INDEX LOC_CITY_IX ON LOCATIONS(CITY ASC)LOGGING;
CREATE INDEX LOC_STATE_PROV_IX ON LOCATIONS(STATE_PROVINCE
ASC)LOGGING;
CREATE INDEX LOC_COUNTRY_IX ON LOCATIONS(COUNTRY_ID
ASC)LOGGING;
CREATE UNIQUE INDEX LOC_ID_PKX ON LOCATIONS(LOCATION_ID ASC);
ALTER TABLE LOCATIONS ADD CONSTRAINT LOC_ID_PK PRIMARY KEY
(LOCATION_ID);

CREATE TABLE REGIONS(
   REGION_ID   NUMBER NOT NULL ,
   REGION_NAME VARCHAR2 (25 BYTE))LOGGING;
CREATE UNIQUE INDEX REG_ID_PKX ON REGIONS(REGION_ID ASC);
ALTER TABLE REGIONS ADD CONSTRAINT REG_ID_PK PRIMARY KEY
(REGION_ID);
```

```
ALTER TABLE COUNTRIES ADD CONSTRAINT COUNTR_REG_FK FOREIGN KEY
(REGION_ID) REFERENCES REGIONS (REGION_ID) NOT DEFERRABLE;

ALTER TABLE DEPARTMENTS ADD CONSTRAINT DEPT_LOC_FK FOREIGN KEY (
LOCATION_ID ) REFERENCES LOCATIONS ( LOCATION_ID ) NOT DEFERRABLE;

ALTER TABLE DEPARTMENTS ADD CONSTRAINT DEPT_MGR_FK FOREIGN KEY
( MANAGER_ID ) REFERENCES EMPLOYEES ( EMPLOYEE_ID ) NOT
DEFERRABLE;

ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_DEPT_FK FOREIGN KEY (
DEPARTMENT_ID ) REFERENCES DEPARTMENTS ( DEPARTMENT_ID ) NOT
DEFERRABLE;

ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_JOB_FK FOREIGN KEY (
JOB_ID ) REFERENCES JOBS ( JOB_ID ) NOT DEFERRABLE;

ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_MANAGER_FK FOREIGN
KEY ( MANAGER_ID ) REFERENCES EMPLOYEES ( EMPLOYEE_ID ) NOT
DEFERRABLE;

ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_DEPT_FK FOREIGN KEY (
DEPARTMENT_ID ) REFERENCES DEPARTMENTS ( DEPARTMENT_ID ) NOT
DEFERRABLE;

ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_EMP_FK FOREIGN KEY (
EMPLOYEE_ID ) REFERENCES EMPLOYEES ( EMPLOYEE_ID ) NOT DEFERRABLE;

ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_JOB_FK FOREIGN KEY (
JOB_ID ) REFERENCES JOBS ( JOB_ID ) NOT DEFERRABLE;

ALTER TABLE LOCATIONS ADD CONSTRAINT LOC_C_ID_FK FOREIGN KEY (
COUNTRY_ID ) REFERENCES COUNTRIES ( COUNTRY_ID ) NOT DEFERRABLE;

CREATE OR REPLACE VIEW EMP_DETAILS_VIEW  AS
SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.country_id,
e.first_name, e.last_name, e.salary, e.commission_pct, d.department_name, j.job_title, l.city,
l.state_province, c.country_name, r.region_name
FROM employees e, departments d, jobs j, locations l, countries c, regions r
WHERE e.department_id = d.department_id
  AND d.location_id = l.location_id
  AND l.country_id = c.country_id
  AND c.region_id = r.region_id
  AND j.job_id = e.job_id
WITH READ ONLY;

CREATE SEQUENCE REGIONS_SEQ INCREMENT BY 1 MAXVALUE 700000
MINVALUE 0 NOCACHE ;

CREATE SEQUENCE COUNTRIES_SEQ INCREMENT BY 1 MAXVALUE 700000
MINVALUE 0 NOCACHE ;
```

```
CREATE SEQUENCE LOCATIONS_SEQ INCREMENT BY 1 MAXVALUE 700000
MINVALUE 0 NOCACHE ;

CREATE SEQUENCE DEPARTMENTS_SEQ INCREMENT BY 1 MAXVALUE 700000
MINVALUE 0 NOCACHE ;

CREATE SEQUENCE EMPLOYEES_SEQ INCREMENT BY 1 MAXVALUE 1000000
MINVALUE 0 NOCACHE ;

CREATE SEQUENCE JOBS_SEQ INCREMENT BY 1 MAXVALUE 700000 MINVALUE
0 NOCACHE ;

CREATE SEQUENCE JOB_HISTORY_SEQ INCREMENT BY 1 MAXVALUE 700000
MINVALUE 0 NOCACHE ;
```

3. pielikums – Tabulu izveide *Microsoft SQL Server* datubāzē

```
CREATE TABLE COUNTRIES(
   COUNTRY_ID   CHAR (2) NOT NULL ,
   COUNTRY_NAME VARCHAR (40) ,
   REGION_ID    NUMERIC (28) ,
   CONSTRAINT COUNTRY_C_ID_PK PRIMARY KEY CLUSTERED (COUNTRY_ID)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON) ON "default" ) ON "default"
GO

CREATE UNIQUE NONCLUSTERED INDEX COUNTRY_C_ID_PKX ON COUNTRIES
( COUNTRY_ID ) ON "default"
GO

CREATE TABLE DEPARTMENTS(
   DEPARTMENT_ID   NUMERIC (4) NOT NULL ,
   DEPARTMENT_NAME VARCHAR (30) NOT NULL ,
   MANAGER_ID      NUMERIC (6) ,
   LOCATION_ID     NUMERIC (4) ,
   CONSTRAINT DEPT_ID_PK PRIMARY KEY CLUSTERED (DEPARTMENT_ID)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON) ON "default" ) ON "default"
GO

CREATE NONCLUSTERED INDEX DEPT_LOCATION_IX ON DEPARTMENTS
( LOCATION_ID ) ON "default"
GO
CREATE UNIQUE NONCLUSTERED INDEX DEPT_ID_PKX ON DEPARTMENTS
( DEPARTMENT_ID ) ON "default"
GO

CREATE TABLE EMPLOYEES(
   EMPLOYEE_ID    NUMERIC (6) NOT NULL ,
   FIRST_NAME     VARCHAR (20) ,
   LAST_NAME      VARCHAR (25) NOT NULL ,
   EMAIL          VARCHAR (25) NOT NULL ,
   PHONE_NUMBER   VARCHAR (20) ,
   HIRE_DATE      DATE NOT NULL ,
   JOB_ID         VARCHAR (10) NOT NULL ,
   SALARY         NUMERIC (8,2) ,
   COMMISSION_PCT NUMERIC (2,2) ,
   MANAGER_ID     NUMERIC (6) ,
   DEPARTMENT_ID  NUMERIC (4) ,
   CONSTRAINT EMP_EMP_ID_PK PRIMARY KEY CLUSTERED (EMPLOYEE_ID)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON ) ON "default" ,
 CONSTRAINT EMP_EMAIL_UK UNIQUE NONCLUSTERED (EMAIL) ON "default" )
 ON "default"
```

```
GO

CREATE NONCLUSTERED INDEX EMP_DEPARTMENT_IX ON EMPLOYEES
( DEPARTMENT_ID ) ON "default"
GO
CREATE NONCLUSTERED INDEX EMP_JOB_IX ON EMPLOYEES
( JOB_ID ) ON "default"
GO
CREATE NONCLUSTERED INDEX EMP_MANAGER_IX ON EMPLOYEES
( MANAGER_ID ) ON "default"
GO
CREATE NONCLUSTERED INDEX EMP_NAME_IX ON EMPLOYEES
( LAST_NAME , FIRST_NAME ) ON "default"
GO
CREATE UNIQUE NONCLUSTERED INDEX EMP_EMP_ID_PKX ON EMPLOYEES
( EMPLOYEE_ID ) ON "default"
GO

CREATE TABLE JOBS(
   JOB_ID     VARCHAR (10) NOT NULL ,
   JOB_TITLE  VARCHAR (35) NOT NULL ,
   MIN_SALARY NUMERIC (6) ,
   MAX_SALARY NUMERIC (6) ,
   CONSTRAINT JOB_ID_PK PRIMARY KEY CLUSTERED (JOB_ID)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON ) ON "default" ) ON "default"
GO

CREATE UNIQUE NONCLUSTERED INDEX JOB_ID_PKX ON JOBS
( JOB_ID ) ON "default"
GO

CREATE TABLE JOB_HISTORY(
   EMPLOYEE_ID   NUMERIC (6) NOT NULL ,
   START_DATE    DATE NOT NULL ,
   END_DATE      DATE NOT NULL ,
   JOB_ID        VARCHAR (10) NOT NULL ,
   DEPARTMENT_ID NUMERIC (4) ,
   CONSTRAINT JHIST_ID_DATE_PK PRIMARY KEY CLUSTERED (EMPLOYEE_ID,
START_DATE)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON ) ON "default" ) ON "default"
GO
ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_DATE_CHECK CHECK (
end_date > start_date )
GO

CREATE NONCLUSTERED INDEX JHIST_JOB_IX ON JOB_HISTORY
( JOB_ID ) ON "default"
GO
```

```
CREATE NONCLUSTERED INDEX JHIST_EMP_IX ON JOB_HISTORY
( EMPLOYEE_ID ) ON "default"
GO
CREATE NONCLUSTERED INDEX JHIST_DEPT_IX ON JOB_HISTORY
( DEPARTMENT_ID ) ON "default"
GO
CREATE UNIQUE NONCLUSTERED INDEX JHIST_ID_DATE_PKX ON
JOB_HISTORY
( EMPLOYEE_ID , START_DATE ) ON "default"
GO

CREATE TABLE LOCATIONS(
   LOCATION_ID    NUMERIC (4) NOT NULL ,
   STREET_ADDRESS VARCHAR (40) ,
   POSTAL_CODE    VARCHAR (12) ,
   CITY           VARCHAR (30) NOT NULL ,
   STATE_PROVINCE VARCHAR (25) ,
   COUNTRY_ID     CHAR (2) ,
   CONSTRAINT LOC_ID_PK PRIMARY KEY CLUSTERED (LOCATION_ID)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON ) ON "default" ) ON "default"
GO

CREATE NONCLUSTERED INDEX LOC_CITY_IX ON LOCATIONS
( CITY ) ON "default"
GO
CREATE NONCLUSTERED INDEX LOC_STATE_PROV_IX ON LOCATIONS
( STATE_PROVINCE ) ON "default"
GO
CREATE NONCLUSTERED INDEX LOC_COUNTRY_IX ON LOCATIONS
( COUNTRY_ID ) ON "default"
GO
CREATE UNIQUE NONCLUSTERED INDEX LOC_ID_PKX ON LOCATIONS
( LOCATION_ID ) ON "default"
GO

CREATE TABLE REGIONS(
   REGION_ID   NUMERIC (28) NOT NULL ,
   REGION_NAME VARCHAR (25) ,
   CONSTRAINT REG_ID_PK PRIMARY KEY CLUSTERED (REGION_ID)
WITH(
   ALLOW_PAGE_LOCKS = ON ,
   ALLOW_ROW_LOCKS  = ON ) ON "default" ) ON "default"
GO

CREATE UNIQUE NONCLUSTERED INDEX REG_ID_PKX ON REGIONS
( REGION_ID ) ON "default"
GO

ALTER TABLE COUNTRIES ADD CONSTRAINT COUNTR_REG_FK FOREIGN KEY
(REGION_ID) REFERENCES REGIONS(REGION_ID) ON DELETE NO ACTION ON
```

```
UPDATE NO ACTION
GO

ALTER TABLE DEPARTMENTS ADD CONSTRAINT DEPT_LOC_FK FOREIGN KEY
(LOCATION_ID) REFERENCES LOCATIONS(LOCATION_ID) ON DELETE NO
ACTION ON UPDATE NO ACTION
GO

ALTER TABLE DEPARTMENTS ADD CONSTRAINT DEPT_MGR_FK FOREIGN KEY
(MANAGER_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID) ON DELETE NO
ACTION ON
UPDATE NO ACTION
GO

ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_DEPT_FK FOREIGN KEY
(DEPARTMENT_ID) REFERENCES DEPARTMENTS(DEPARTMENT_ID) ON DELETE
NO ACTION ON UPDATE NO ACTION
GO

ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_JOB_FK FOREIGN KEY
(JOB_ID) REFERENCES JOBS(JOB_ID) ON DELETE NO ACTION ON UPDATE NO
ACTION
GO

ALTER TABLE EMPLOYEES ADD CONSTRAINT EMP_MANAGER_FK FOREIGN
KEY
(MANAGER_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID) ON DELETE NO
ACTION ON
UPDATE NO ACTION
GO

ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_DEPT_FK FOREIGN KEY
(DEPARTMENT_ID) REFERENCES DEPARTMENTS(DEPARTMENT_ID) ON DELETE
NO ACTION ON UPDATE NO ACTION
GO

ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_EMP_FK FOREIGN KEY
(EMPLOYEE_ID) REFERENCES EMPLOYEES (EMPLOYEE_ID) ON DELETE NO
ACTION ON UPDATE NO ACTION
GO

ALTER TABLE JOB_HISTORY ADD CONSTRAINT JHIST_JOB_FK FOREIGN KEY
(JOB_ID) REFERENCES JOBS(JOB_ID) ON DELETE NO ACTION ON UPDATE NO
ACTION
GO

ALTER TABLE LOCATIONS ADD CONSTRAINT LOC_C_ID_FK FOREIGN KEY
(COUNTRY_ID) REFERENCES COUNTRIES(COUNTRY_ID) ON DELETE NO
ACTION ON
UPDATE NO ACTION
GO
```

```
CREATE VIEW EMP_DETAILS_VIEW  AS
SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.country_id,
e.first_name, e.last_name, e.salary, e.commission_pct, d.department_name, j.job_title, l.city,
l.state_province, c.country_name, r.region_name FROM employees e, departments d, jobs j,
locations l, countries c, regions r WHERE e.department_id = d.department_id AND
d.location_id = l.location_id AND l.country_id = c.country_id AND c.region_id = r.region_id
AND j.job_id = e.job_id
GO

CREATE SEQUENCE HR.REGIONS_SEQ START WITH 1 INCREMENT BY 1
MINVALUE 0 MAXVALUE 700000 NO CACHE;
GO

CREATE SEQUENCE HR.COUNTRIES_SEQ START WITH 1 INCREMENT BY 1
MINVALUE 0 MAXVALUE 700000 NO CACHE;
GO

CREATE SEQUENCE HR.LOCATIONS_SEQ START WITH 1 INCREMENT BY 1
MINVALUE 0 MAXVALUE 700000 NO CACHE;
GO

CREATE SEQUENCE HR.DEPARTMENTS_SEQ START WITH 1 INCREMENT BY 1
MINVALUE 0 MAXVALUE 700000 NO CACHE;
GO

CREATE SEQUENCE HR.EMPLOYEES_SEQ START WITH 1 INCREMENT BY 1
MINVALUE 0 MAXVALUE 1000000 NO CACHE;
GO

CREATE SEQUENCE HR.JOBS_SEQ START WITH 1 INCREMENT BY 1 MINVALUE
0 MAXVALUE 700000 NOCACHE;
GO

CREATE SEQUENCE HR.JOB_HISTORY_SEQ START WITH 1 INCREMENT BY 1
MINVALUE 0 MAXVALUE 700000 NO CACHE;
GO
```

4. pielikums – HR shēmas tabulu apraksts

4.1. tabula – **HR shēmas tabulu apraksts** [21]

Tabulas nosaukums	Kolonnas nosaukums	Datu tips	Specifiski uzstādījumi
REGIONS	REGION_ID	NUMBER	Primārā atslēga, NOT NULL
	REGION_NAME	VARCHAR2(25)	
COUNTRIES	COUNTRY_ID	CHAR(2)	Primārā atslēga, NOT NULL
	COUNTRY_NAME	VARCHAR2(40)	
	REGION_ID	NUMBER	Sekundārā atslēga
LOCATIONS	LOCATION_ID	NUMBER(4)	Primārā atslēga, NOT NULL
	STREET_ADDRESS	VARCHAR2(40)	
	POSTAL_CODE	VARCHAR2(12)	
	CITY	VARCHAR2(30)	NOT NULL
	STATE_PROVINCE	VARCHAR2(25)	
	COUNTRY_ID	CHAR(2)	Sekundārā atslēga
DEPARTMENTS	DEPARTMENT_ID	NUMBER(4)	Primārā atslēga, NOT NULL
	DEPARTMENT_NAME	VARCHAR2(30)	NOT NULL
	MANAGER_ID	NUMBER(6)	Sekundārā atslēga
	LOCATION_ID	NUMBER(4)	Sekundārā atslēga
JOBS	JOB_ID	VARCHAR2(10)	Primārā atslēga, NOT NULL
	JOB_TITLE	VARCHAR2(35)	NOT NULL
	MIN_SALARY	NUMBER(6)	
	MAX_SALARY	NUMBER(6)	
JOB_HISTORY	EMPLOYEE_ID	NUMBER(6)	Salikta primārā atslēga, sekundārā atslēga, NOT NULL
	START_DATE	DATE	Salikta primārā atslēga, NOT NULL

	END_DATE	DATE	NOT NULL
	JOB_ID	VARCHAR2(10)	Sekundārā atslēga, NOT NULL
	DEPARTMENT_ID	NUMBER(4)	Sekundārā atslēga
EMPLOYEES	EMPLOYEE_ID	NUMBER(6)	Primārā atslēga, NOT NULL
	FIRST_NAME	VARCHAR2(20)	
	LAST_NAME	VARCHAR2(25)	NOT NULL
	EMAIL	VARCHAR2(20)	NOT NULL, unikāls
	PHONE_NUMBER	VARCHAR2(20)	
	HIRE_DATE	DATE	NOT NULL
	JOB_ID	VARCHAR2(10)	Sekundārā atslēga, NOT NULL
	SALARY	NUMBER(8, 2)	
	COMMISSION_PCT	NUMBER(2, 2)	
	MANAGER_ID	NUMBER(6)	Sekundārā atslēga
	DEPARTMENT_ID	NUMBER(4)	Sekundārā atslēga

5. pielikums – Sākuma datu ievades komandas

```
INSERT INTO regions VALUES ( 1, 'Europe' );
INSERT INTO regions VALUES ( 2, 'Americas' );
INSERT INTO regions VALUES ( 3, 'Asia' );
INSERT INTO regions VALUES ( 4, 'Middle East and Africa' );

INSERT INTO countries VALUES ( 'IT', 'Italy', 1 );
INSERT INTO countries VALUES ( 'JP', 'Japan', 3 );
INSERT INTO countries VALUES ( 'US', 'United States of America', 2 );
INSERT INTO countries VALUES ( 'CA', 'Canada', 2 );
INSERT INTO locations VALUES ( 1000 , '1297 Via Cola di Rie', '00989', 'Roma', NULL,
'IT');
INSERT INTO locations VALUES ( 1100 , '93091 Calle della Testa', '10934', 'Venice',
NULL, 'IT');
INSERT INTO locations VALUES ( 1200 , '2017 Shinjuku-ku', '1689', 'Tokyo', 'Tokyo
Prefecture', 'JP');
INSERT INTO locations VALUES ( 1300 , '9450 Kamiya-cho', '6823', 'Hiroshima', NULL,
'JP');
INSERT INTO locations VALUES ( 1400 , '2014 Jabberwocky Rd', '26192' , 'Southlake',
'Texas', 'US' );

INSERT INTO departments VALUES ( 10 , 'Administration', 200, 1700);
INSERT INTO departments VALUES ( 20 , 'Marketing', 201, 1800);
INSERT INTO departments VALUES ( 30 , 'Purchasing', 114, 1700);
INSERT INTO departments VALUES ( 40 , 'Human Resources', 203, 2400);
INSERT INTO jobs VALUES ( 'AD_PRES', 'President', 20000, 40000);
INSERT INTO jobs VALUES ( 'AD_VP', 'Administration Vice President', 15000, 30000);
INSERT INTO jobs VALUES ( 'AD_ASST', 'Administration Assistant', 3000, 6000);
INSERT INTO jobs VALUES ( 'FI_MGR', 'Finance Manager', 8200, 16000);
INSERT INTO employees VALUES ( 100 , 'Steven', 'King', 'SKING', '515.123.4567',
TO_DATE('17-JUN-1987', 'dd-MON-yyyy'), 'AD_PRES', 24000, NULL, NULL, 90);
INSERT INTO employees VALUES ( 101 , 'Neena', 'Kochhar', 'NKOCHHAR',
'515.123.4568', TO_DATE('21-SEP-1989', 'dd-MON-yyyy'), 'AD_VP', 17000, NULL, 100,
90);
INSERT INTO employees VALUES ( 102 , 'Lex', 'De Haan', 'LDEHAAN', '515.123.4569',
TO_DATE('13-JAN-1993', 'dd-MON-yyyy'), 'AD_VP', 17000, NULL, 100, 90);
INSERT INTO employees VALUES ( 103 , 'Alexander', 'Hunold', 'AHUNOLD',
'590.423.4567', TO_DATE('03-JAN-1990', 'dd-MON-yyyy'), 'IT_PROG', 9000, NULL, 102,
60);

INSERT INTO job_history VALUES (102, TO_DATE('13-JAN-1993', 'dd-MON-yyyy'),
TO_DATE('24-JUL-1998', 'dd-MON-yyyy'), 'IT_PROG', 60);
INSERT INTO job_history VALUES (101, TO_DATE('21-SEP-1989', 'dd-MON-yyyy'),
TO_DATE('27-OCT-1993', 'dd-MON-yyyy'), 'AC_ACCOUNT', 110);
INSERT INTO job_history VALUES (101, TO_DATE('28-OCT-1993', 'dd-MON-yyyy'),
TO_DATE('15-MAR-1997', 'dd-MON-yyyy'), 'AC_MGR', 110);
INSERT INTO job_history VALUES (201, TO_DATE('17-FEB-1996', 'dd-MON-yyyy'),
TO_DATE('19-DEC-1999', 'dd-MON-yyyy'), 'MK_REP', 20);
```

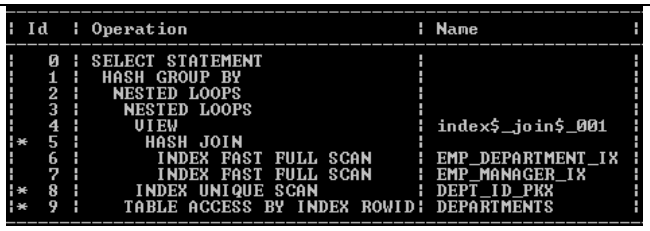
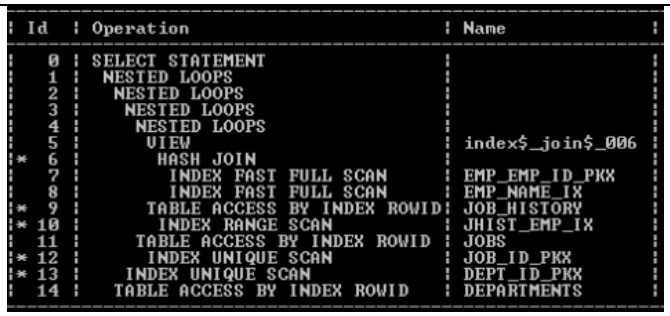
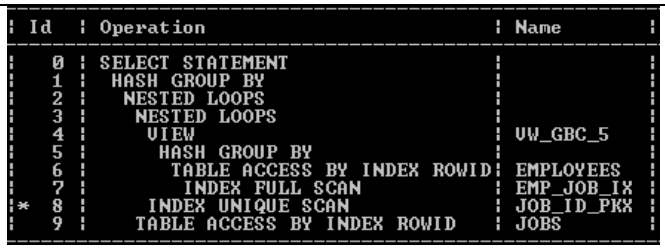
6. pielikums – Trīs izmantotie SQL pieprasījumi praktiskajā daļā

```
-- 1. Oracle
SELECT DEPARTMENT_NAME, COUNT(*) FROM EMPLOYEES
NATURAL JOIN DEPARTMENTS GROUP BY DEPARTMENT_NAME;

-- 1. Microsoft SQL Server
SELECT D.DEPARTMENT_NAME, COUNT(*)
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
GROUP BY D.DEPARTMENT_NAME;

-- 2. Oracle
SELECT JOB_TITLE, DEPARTMENT_NAME, LAST_NAME, START_DATE
FROM JOB_HISTORY JOIN JOBS USING (JOB_ID) JOIN DEPARTMENTS
USING (DEPARTMENT_ID) JOIN  EMPLOYEES USING (EMPLOYEE_ID)
WHERE TO_CHAR(START_DATE,'YYYY') BETWEEN 1990 AND 2000;

-- 2. Microsoft SQL Server
SELECT J.JOB_TITLE, D.DEPARTMENT_NAME, E.LAST_NAME, JH.START_DATE
FROM JOB_HISTORY JH
JOIN JOBS J ON JH.JOB_ID = J.JOB_ID
JOIN DEPARTMENTS D ON D.DEPARTMENT_ID = JH.DEPARTMENT_ID
JOIN EMPLOYEES E ON E.EMPLOYEE_ID = JH.EMPLOYEE_ID
WHERE JH.START_DATE BETWEEN '1990-01-01' AND '2000-01-01';

-- 3. Oracle
SELECT JOB_TITLE, AVG(SALARY) FROM EMPLOYEES
NATURAL JOIN JOBS GROUP BY JOB_TITLE;

-- 3. Microsoft SQL Server
SELECT J.JOB_TITLE, AVG(E.SALARY)
FROM EMPLOYEES E
JOIN JOBS J ON J.JOB_ID = E.JOB_ID
GROUP BY J.JOB_TITLE;
```

7. pielikums – *Oracle 11g* praktiskās daļas rezultāti

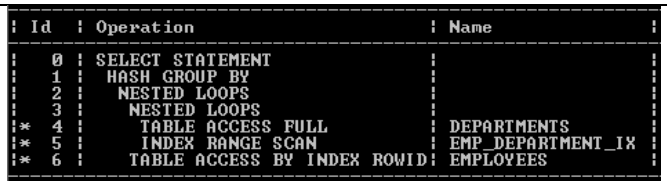
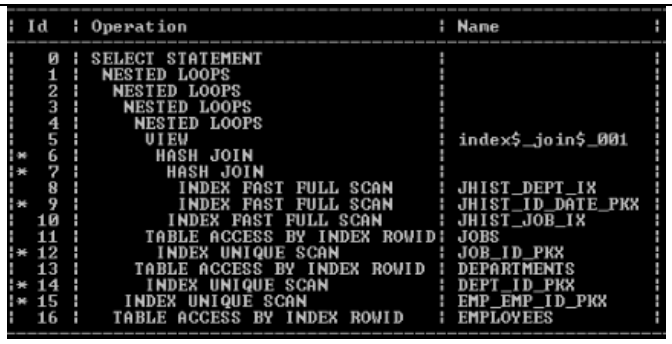
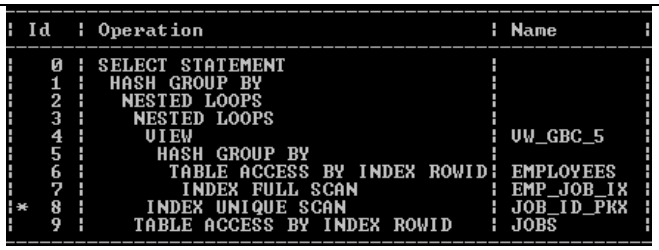
7.1. tabula – ***Oracle 11g*** **praktiskās daļas 1. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 9.42 milisekundes, izmantojot komandu
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (ownname => 'HR', estimate_percent => 100, options => 'GATHER');</td></tr>
<tr><td>2.</td><td>Sākuma dati</td><td colspan="2">Ievadīti sākuma dati HR shēmas tabulās:
REGIONS – 4, COUNTRIES – 25, LOCATIONS – 23,
DEPARTMENTS – 27, JOBS – 19, JOB_HISTORY – 10,
EMPLOYEES – 107.</td></tr>
<tr><td>3.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 100% procentuālo novirzi, jo pirmo reizi pēc datubāzes uzstādīšanas tiek veikta statistiku vākšana. Izmantotā komanda:
SELECT COUNT(*) FROM HR.table_name;
SELECT num_rows FROM dba_tables WHERE table_name ='table_name' AND owner = 'HR';</td></tr>
<tr><td rowspan="3">4.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	VIEW	index$_join$_001
* 5	HASH JOIN	
6	INDEX FAST FULL SCAN	EMP_DEPARTMENT_IX
7	INDEX FAST FULL SCAN	EMP_MANAGER_IX
* 8	INDEX UNIQUE SCAN	DEPT_ID_PKX
* 9	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
</pre>
</td></tr>
<tr><td>2.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	NESTED LOOPS	
5	VIEW	index$_join$_006
* 6	HASH JOIN	
7	INDEX FAST FULL SCAN	EMP_EMP_ID_PKX
8	INDEX FAST FULL SCAN	EMP_NAME_IX
* 9	TABLE ACCESS BY INDEX ROWID	JOB_HISTORY
* 10	INDEX RANGE SCAN	JHIST_EMP_IX
11	TABLE ACCESS BY INDEX ROWID	JOBS
* 12	INDEX UNIQUE SCAN	JOB_ID_PKX
* 13	INDEX UNIQUE SCAN	DEPT_ID_PKX
14	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
</pre>
</td></tr>
<tr><td>3.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	TABLE ACCESS FULL	JOBS
* 5	INDEX RANGE SCAN	EMP_JOB_IX
6	TABLE ACCESS BY INDEX ROWID	EMPLOYEES
</pre>
</td></tr>
<tr><td>5.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 2.79 milisekundes, izmantojot komandu
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (ownname => 'HR', estimate_percent => 100, options =></td></tr>
</table>

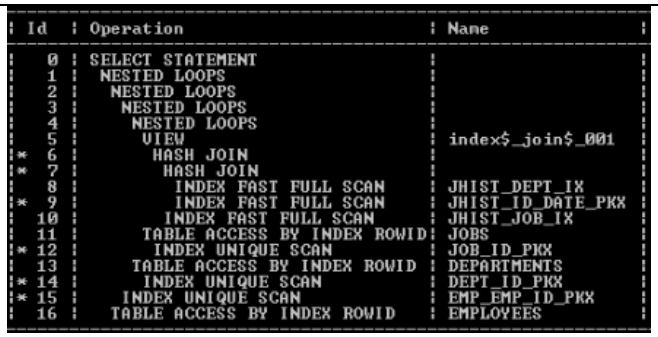
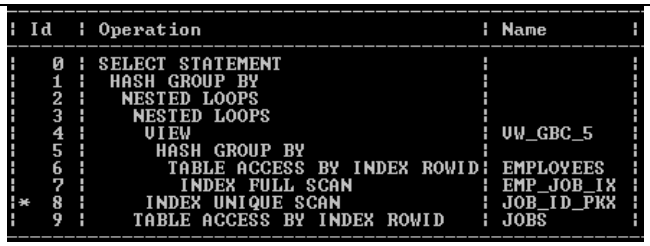
<table>
<tr><td></td><td></td><td colspan="2">'GATHER');</td></tr>
<tr><td>6.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
SELECT COUNT(*) FROM HR.table_name;
SELECT num_rows FROM dba_tables WHERE table_name ='table_name' AND owner = 'HR';</td></tr>
<tr><td rowspan="3">7.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
</table>

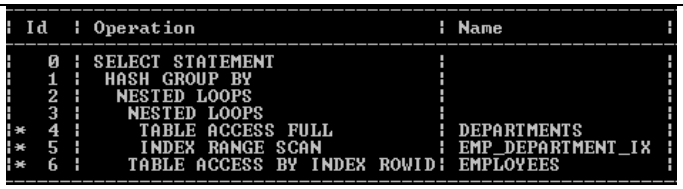
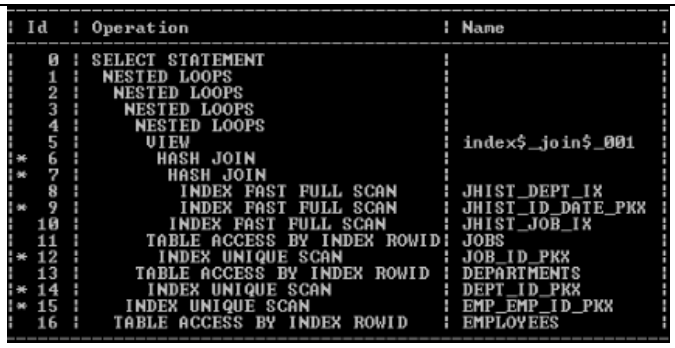
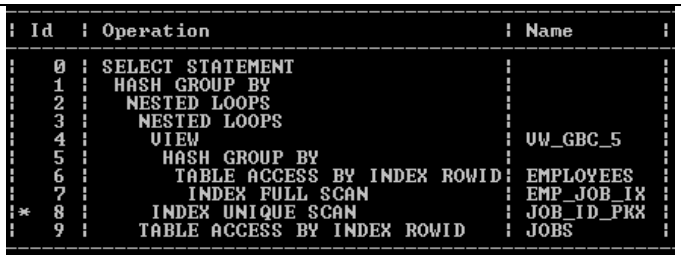
7.2. tabula – ***Oracle 11g*** **praktiskās daļas 2. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas tabulā EMPLOYEES, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td rowspan="3">2.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	VIEW	index$_join$_001
* 5	HASH JOIN	
6	INDEX FAST FULL SCAN	EMP_DEPARTMENT_IX
7	INDEX FAST FULL SCAN	EMP_MANAGER_IX
* 8	INDEX UNIQUE SCAN	DEPT_ID_PKX
* 9	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
</pre>
</td></tr>
<tr><td>2.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	NESTED LOOPS	
5	VIEW	index$_join$_006
* 6	HASH JOIN	
7	INDEX FAST FULL SCAN	EMP_EMP_ID_PKX
8	INDEX FAST FULL SCAN	EMP_NAME_IX
* 9	TABLE ACCESS BY INDEX ROWID	JOB_HISTORY
* 10	INDEX RANGE SCAN	JHIST_EMP_IX
11	TABLE ACCESS BY INDEX ROWID	JOBS
* 12	INDEX UNIQUE SCAN	JOB_ID_PKX
* 13	INDEX UNIQUE SCAN	DEPT_ID_PKX
14	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
</pre>
</td></tr>
<tr><td>3.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	VIEW	VW_GBC_5
5	HASH GROUP BY	
6	TABLE ACCESS BY INDEX ROWID	EMPLOYEES
7	INDEX FULL SCAN	EMP_JOB_IX
* 8	INDEX UNIQUE SCAN	JOB_ID_PKX
9	TABLE ACCESS BY INDEX ROWID	JOBS
</pre>
</td></tr>
<tr><td>3.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
99.89%</td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 7.67 milisekundes, izmantojot komandu
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (ownname => 'HR', estimate_percent => 100, options => 'GATHER');</td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
SELECT COUNT(*) FROM HR.table_name;
SELECT num_rows FROM dba_tables WHERE table_name ='table_name' AND owner = 'HR';</td></tr>
</table>

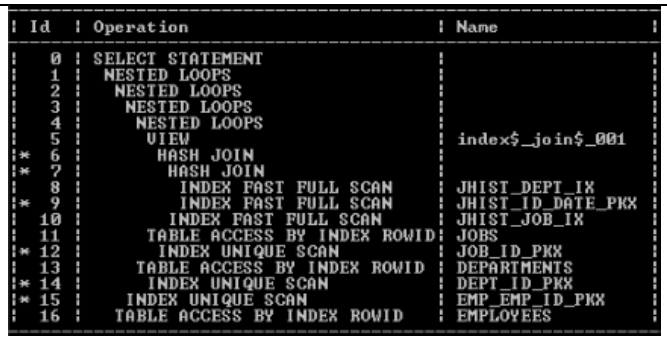
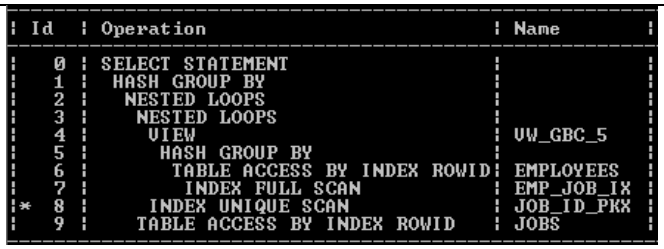
6.	SQL pieprasījumu izpildes plāni	1.	
		2.	
		3.	

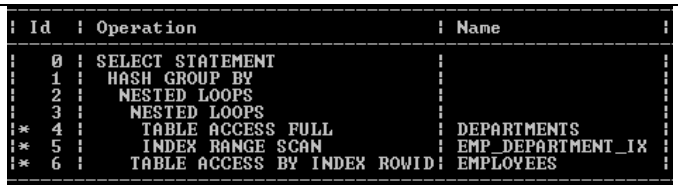
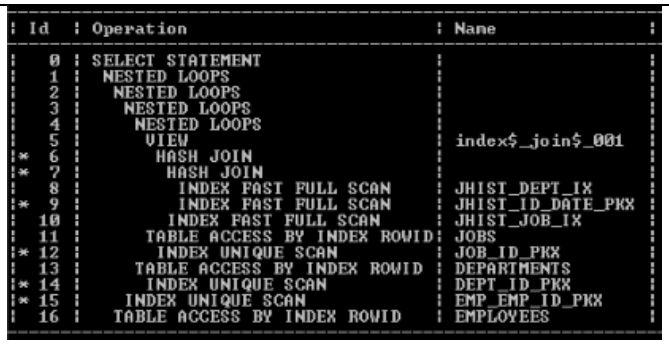
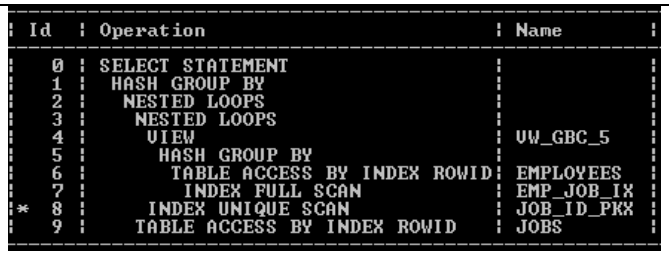
7.3. tabula – ***Oracle 11g*** **praktiskās daļas 3. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas tabulā EMPLOYEES, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td rowspan="3">2.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
<tr><td>3.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
49.99%</td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 56.84 milisekundes, izmantojot komandu
<code>EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (ownname => 'HR', estimate_percent => 100, options => 'GATHER');</code></td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
<code>SELECT COUNT(*) FROM HR.table_name;</code>
<code>SELECT num_rows FROM dba_tables WHERE table_name ='table_name' AND owner = 'HR';</code></td></tr>
</table>

6.	SQL pieprasījumu izpildes plāni	1.	
		2.	
		3.	

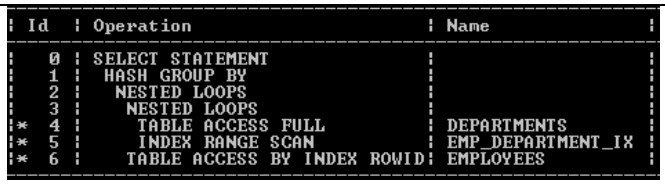
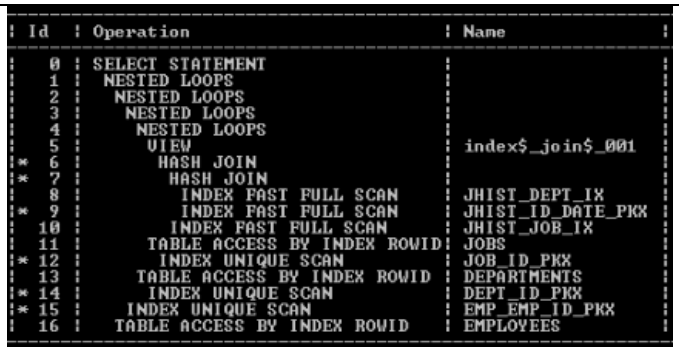
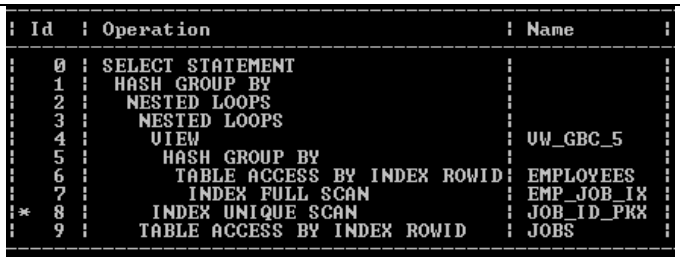
7.4. tabula – ***Oracle 11g*** **praktiskās daļas 4. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas tabulā EMPLOYEES, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td rowspan="3">2.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td> 
</td></tr>
<tr><td>2.</td><td> 
</td></tr>
<tr><td>3.</td><td> 
</td></tr>
<tr><td>3.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
33.32%</td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 22.74 milisekundes, izmantojot komandu
<code>EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (ownname => 'HR', estimate_percent => 100, options => 'GATHER');</code></td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
<code>SELECT COUNT(*) FROM HR.table_name;</code>
<code>SELECT num_rows FROM dba_tables WHERE table_name ='table_name' AND owner = 'HR';</code></td></tr>
</table>

6.	SQL pieprasījumu izpildes plāni	1.	
		2.	
		3.	

7.5. tabula – ***Oracle 11g*** **praktiskās daļas 5. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas tabulā EMPLOYEES, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td rowspan="3">2.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
* 4	TABLE ACCESS FULL	DEPARTMENTS
* 5	INDEX RANGE SCAN	EMP_DEPARTMENT_IX
* 6	TABLE ACCESS BY INDEX ROWID	EMPLOYEES
</pre>
</td></tr>
<tr><td>2.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	NESTED LOOPS	
5	VIEW	index$_join$_001
* 6	HASH JOIN	
* 7	HASH JOIN	
8	INDEX FAST FULL SCAN	JHIST_DEPT_IX
* 9	INDEX FAST FULL SCAN	JHIST_ID_DATE_PKX
10	INDEX FAST FULL SCAN	JHIST_JOB_IX
11	TABLE ACCESS BY INDEX ROWID	JOBS
* 12	INDEX UNIQUE SCAN	JOB_ID_PKX
13	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
* 14	INDEX UNIQUE SCAN	DEPT_ID_PKX
* 15	INDEX UNIQUE SCAN	EMP_EMP_ID_PKX
16	TABLE ACCESS BY INDEX ROWID	EMPLOYEES
</pre>
</td></tr>
<tr><td>3.</td><td>
<pre>
Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	VIEW	VW_GBC_5
5	HASH GROUP BY	
6	TABLE ACCESS BY INDEX ROWID	EMPLOYEES
7	INDEX FULL SCAN	EMP_JOB_IX
* 8	INDEX UNIQUE SCAN	JOB_ID_PKX
9	TABLE ACCESS BY INDEX ROWID	JOBS
</pre>
</td></tr>
<tr><td>3.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
69.96%</td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 42.90 milisekundes, izmantojot komandu
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (ownname => 'HR', estimate_percent => 100, options => 'GATHER');</td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
SELECT COUNT(*) FROM HR.table_name;
SELECT num_rows FROM dba_tables WHERE table_name ='table_name' AND owner = 'HR';</td></tr>
</table>

6.	SQL pieprasījumu izpildes plāni	1.	
		2.	
		3.	

8. pielikums – *Microsoft SQL Server 2012* praktiskās daļas rezultāti

8.1. tabula – ***Microsoft SQL Server*** **praktiskās daļas 1. soļa rezultāti**

NPK	Darbības nosaukums	Rezultāti
1.	Statistiku vākšana	Izpildes laiks ir 10.02 milisekundes. Tiek izmantota komanda katrai tabulai ar atbilstošu statistikas nosaukumu: `CREATE STATISTICS stat1 ON dbo.table_name (column_name, column_name) WITH SAMPLE 100 PERCENT;`
2.	Sākuma dati	Ievadīti sākuma dati HR shēmas tabulās: REGIONS – 4, COUNTRIES – 25, LOCATIONS – 23, DEPARTMENTS – 27, JOBS – 19, JOB_HISTORY – 10, EMPLOYEES – 107.

3.	Procentuālās novirzes	Visas tabulas un shēmas ir ar 100% procentuālo novirzi, jo pirmo reizi pēc datubāzes uzstādīšanas tiek veikta statistiku vākšana. Izmantotā komanda:
		`SELECT COUNT(*) FROM dbo.table_name;`
		`DBCC SHOW_STATISTICS ( "dbo.table_name" , stat1) WITH stat_header;`

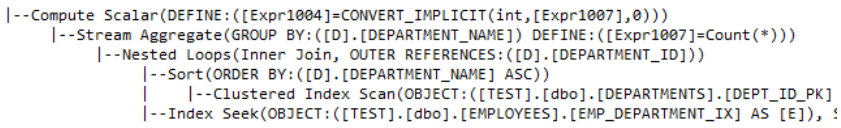
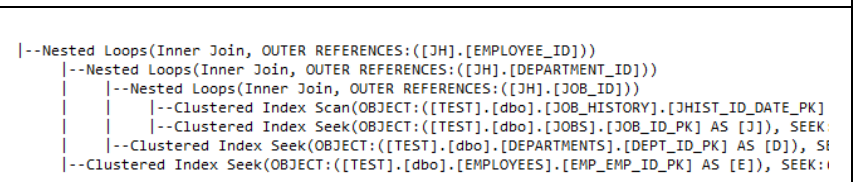
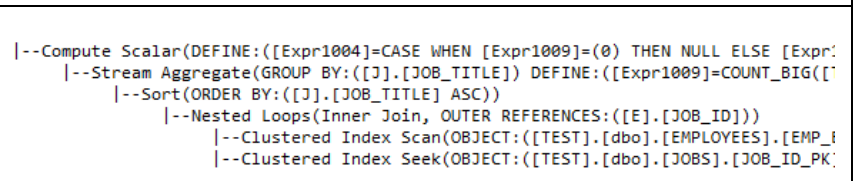
4.	SQL pieprasījumu izpildes plāni	1.
		2.
		3.
5.	Statistiku vākšana	Izpildes laiks ir 4.56 milisekundes, izmantojot komandu katrai tabulai HR shēmā: `UPDATE STATISTICS dbo.table_name;`

6.	Procentuālās novirzes	Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
		`SELECT COUNT(*) FROM dbo.table_name;`
		`DBCC SHOW_STATISTICS ( "dbo.table_name" , stat1) WITH`

```
|--Compute Scalar(DEFINE:([Expr1004]=CONVERT_IMPLICIT(int,[Expr1007],0)))
     |--Stream Aggregate(GROUP BY:([D].[DEPARTMENT_NAME]) DEFINE:([Expr1007]=Count(*)))
          |--Nested Loops(Inner Join, OUTER REFERENCES:([D].[DEPARTMENT_ID]))
               |--Sort(ORDER BY:([D].[DEPARTMENT_NAME] ASC))
               |    |--Clustered Index Scan(OBJECT:([TEST].[dbo].[DEPARTMENTS].[DEPT_ID_PK]
               |--Index Seek(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_DEPARTMENT_IX] AS [E]),
```

```
|--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[EMPLOYEE_ID]))
     |--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[DEPARTMENT_ID]))
     |    |--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[JOB_ID]))
     |    |    |--Clustered Index Scan(OBJECT:([TEST].[dbo].[JOB_HISTORY].[JHIST_ID_DATE_PK]
     |    |    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[JOBS].[JOB_ID_PK] AS [J]), SEEK
     |    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[DEPARTMENTS].[DEPT_ID_PK] AS [D]), S
     |--Clustered Index Seek(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_EMP_ID_PK] AS [E]), SEEK:
```

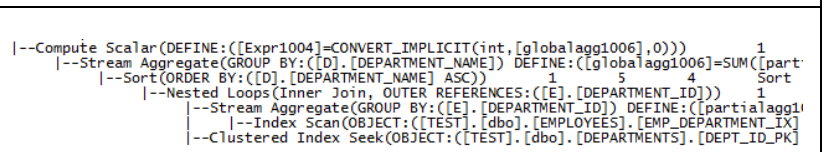
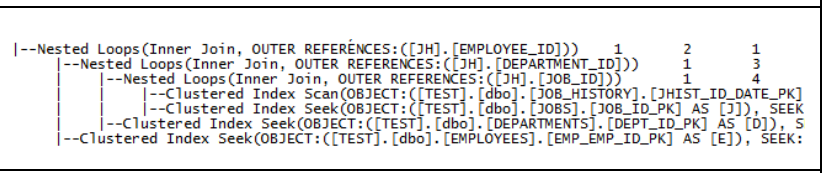
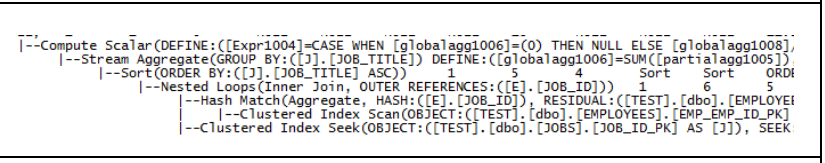
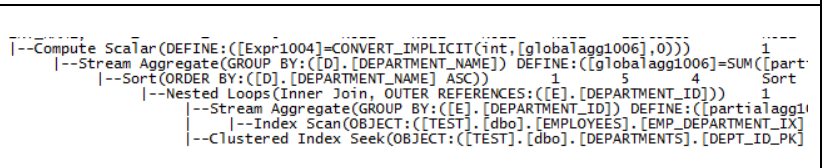
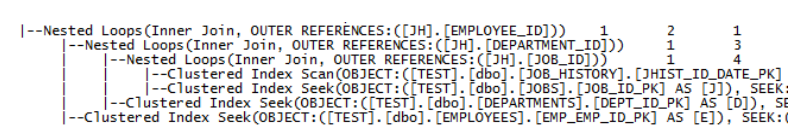
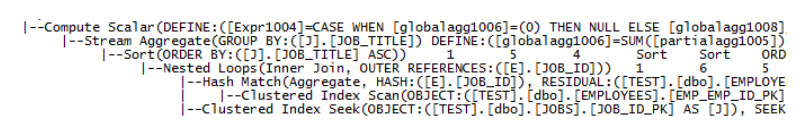
```
|--Compute Scalar(DEFINE:([Expr1004]=CASE WHEN [Expr1009]=(0) THEN NULL ELSE [Expr1010]/CONVI
     |--Stream Aggregate(GROUP BY:([J].[JOB_TITLE]) DEFINE:([Expr1009]=COUNT_BIG([TEST].[dbo
          |--Sort(ORDER BY:([J].[JOB_TITLE] ASC))
               |--Nested Loops(Inner Join, OUTER REFERENCES:([E].[JOB_ID]))
                    |--Clustered Index Scan(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_EMP_ID_PK]
                    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[JOBS].[JOB_ID_PK] AS [J]),
```

		`stat_header;`	
7.	SQL pieprasījumu izpildes plāni	1.	
		2.	
		3.	

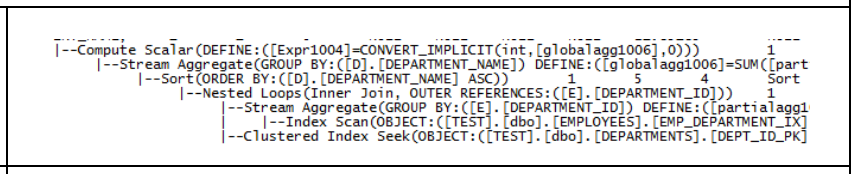
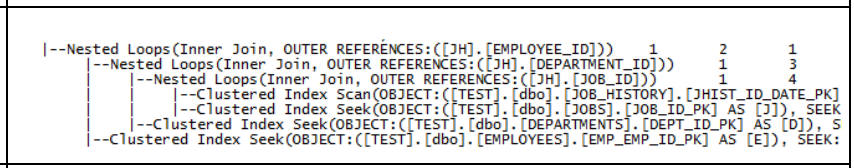
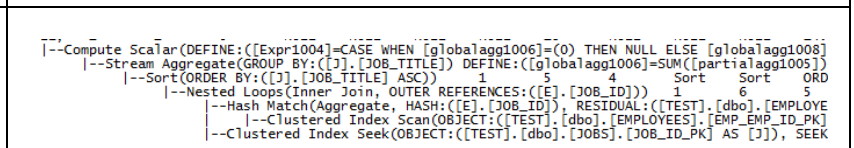
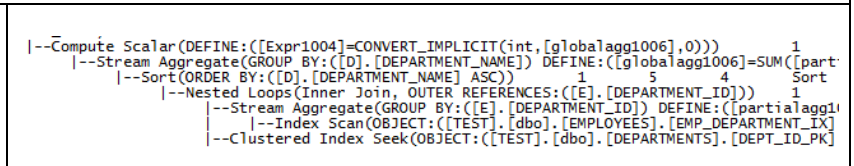
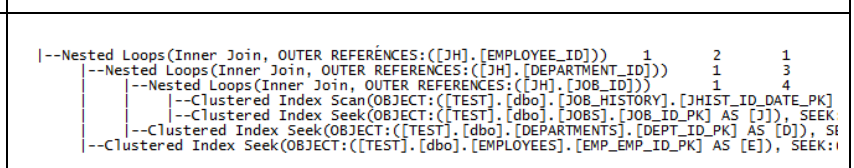
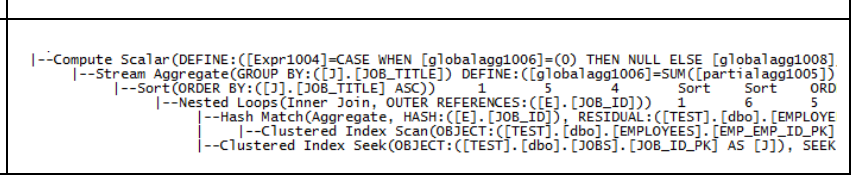
8.2. tabula – ***Microsoft SQL Server*** **praktiskās daļas 2. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas EMPLOYEES tabulā, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td>2.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
99.89%</td></tr>
<tr><td rowspan="3">3.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td><pre>|--Compute Scalar(DEFINE:([Expr1004]=CONVERT_IMPLICIT(int,[globalagg1006],0)))
     |--Stream Aggregate(GROUP BY:([D].[DEPARTMENT_NAME]) DEFINE:([globalagg1006]=SUM([part
          |--Sort(ORDER BY:([D].[DEPARTMENT_NAME] ASC))
               |--Nested Loops(Inner Join, OUTER REFERENCES:([E].[DEPARTMENT_ID]))
                    |--Stream Aggregate(GROUP BY:([E].[DEPARTMENT_ID]) DEFINE:([partialagg1(
                    | |--Index Scan(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_DEPARTMENT_IX]
                    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[DEPARTMENTS].[DEPT_ID_PK]</pre></td></tr>
<tr><td>2.</td><td><pre>|--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[EMPLOYEE_ID]))
     |--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[DEPARTMENT_ID]))
     | |--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[JOB_ID]))
     | | |--Clustered Index Scan(OBJECT:([TEST].[dbo].[JOB_HISTORY].[JHIST_ID_DATE_PK]
     | | |--Clustered Index Seek(OBJECT:([TEST].[dbo].[JOBS].[JOB_ID_PK] AS [J]), SEEK
     | |--Clustered Index Seek(OBJECT:([TEST].[dbo].[DEPARTMENTS].[DEPT_ID_PK] AS [D]), SI
     |--Clustered Index Seek(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_EMP_ID_PK] AS [E]), SEEK:</pre></td></tr>
<tr><td>3.</td><td><pre>|--Compute Scalar(DEFINE:([Expr1004]=CASE WHEN [globalagg1006]=(0) THEN NULL ELSE [globalagg1008]
     |--Stream Aggregate(GROUP BY:([J].[JOB_TITLE]) DEFINE:([globalagg1006]=SUM([partialagg1005])
          |--Sort(ORDER BY:([J].[JOB_TITLE] ASC))
               |--Nested Loops(Inner Join, OUTER REFERENCES:([E].[JOB_ID]))
                    |--Hash Match(Aggregate, HASH:([E].[JOB_ID]), RESIDUAL:([TEST].[dbo].[EMPLOYEE
                    | |--Clustered Index Scan(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_EMP_ID_PK]
                    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[JOBS].[JOB_ID_PK] AS [J]), SEEK</pre></td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 6.54 milisekundes, izmantojot komandu katrai tabulai HR shēmā:
<pre>UPDATE STATISTICS dbo.table_name;</pre></td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
<pre>SELECT COUNT(*) FROM dbo.table_name;
DBCC SHOW_STATISTICS ("dbo.table_name" , stat1) WITH stat_header;</pre></td></tr>
<tr><td rowspan="3">6.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td><pre>|--Compute Scalar(DEFINE:([Expr1004]=CONVERT_IMPLICIT(int,[globalagg1006],0)))
     |--Stream Aggregate(GROUP BY:([D].[DEPARTMENT_NAME]) DEFINE:([globalagg1006]=SUM([part
          |--Sort(ORDER BY:([D].[DEPARTMENT_NAME] ASC))
               |--Nested Loops(Inner Join, OUTER REFERENCES:([E].[DEPARTMENT_ID]))
                    |--Stream Aggregate(GROUP BY:([E].[DEPARTMENT_ID]) DEFINE:([partialagg1(
                    | |--Index Scan(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_DEPARTMENT_IX]
                    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[DEPARTMENTS].[DEPT_ID_PK]</pre></td></tr>
<tr><td>2.</td><td><pre>|--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[EMPLOYEE_ID]))
     |--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[DEPARTMENT_ID]))
     | |--Nested Loops(Inner Join, OUTER REFERENCES:([JH].[JOB_ID]))
     | | |--Clustered Index Scan(OBJECT:([TEST].[dbo].[JOB_HISTORY].[JHIST_ID_DATE_PK]
     | | |--Clustered Index Seek(OBJECT:([TEST].[dbo].[JOBS].[JOB_ID_PK] AS [J]), SEEK
     | |--Clustered Index Seek(OBJECT:([TEST].[dbo].[DEPARTMENTS].[DEPT_ID_PK] AS [D]), SI
     |--Clustered Index Seek(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_EMP_ID_PK] AS [E]), SEEK:</pre></td></tr>
<tr><td>3.</td><td><pre>|--Compute Scalar(DEFINE:([Expr1004]=CASE WHEN [globalagg1006]=(0) THEN NULL ELSE [globalagg1008]
     |--Stream Aggregate(GROUP BY:([J].[JOB_TITLE]) DEFINE:([globalagg1006]=SUM([partialagg1005])
          |--Sort(ORDER BY:([J].[JOB_TITLE] ASC))
               |--Nested Loops(Inner Join, OUTER REFERENCES:([E].[JOB_ID]))
                    |--Hash Match(Aggregate, HASH:([E].[JOB_ID]), RESIDUAL:([TEST].[dbo].[EMPLOYE
                    | |--Clustered Index Scan(OBJECT:([TEST].[dbo].[EMPLOYEES].[EMP_EMP_ID_PK]
                    |--Clustered Index Seek(OBJECT:([TEST].[dbo].[JOBS].[JOB_ID_PK] AS [J]), SEEK</pre></td></tr>
</table>

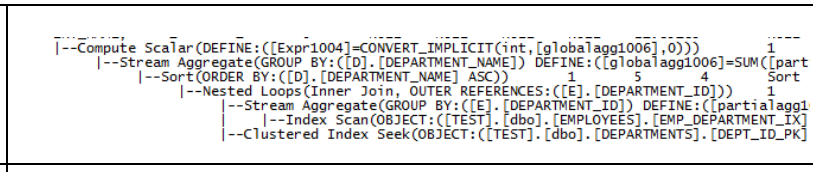
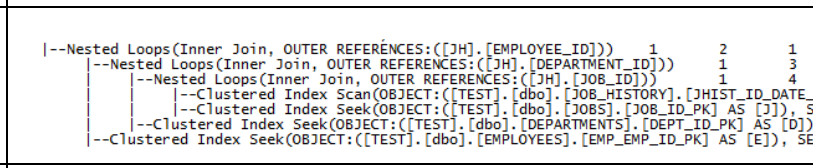
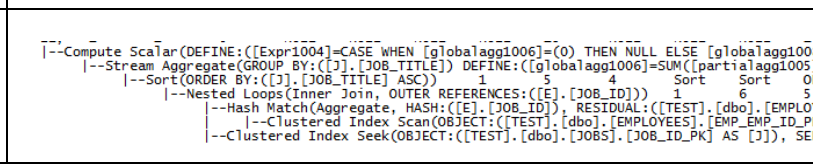
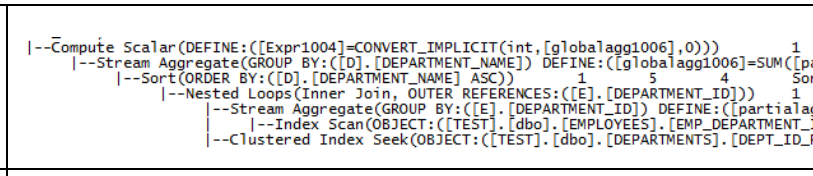
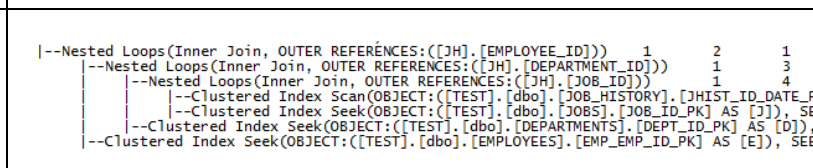
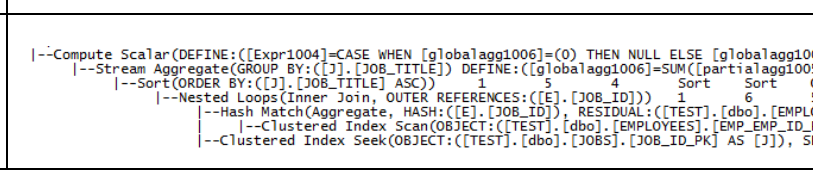
8.3. tabula – ***Microsoft SQL Server*** **praktiskās daļas 3. soļa rezultāti**

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas EMPLOYEES tabulā, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td>2.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
98.75%</td></tr>
<tr><td rowspan="3">3.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 7.03 milisekundes, izmantojot komandu katrai tabulai HR shēmā:
<code>UPDATE STATISTICS dbo.table_name;</code></td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
<code>SELECT COUNT(*) FROM dbo.table_name;</code>
<code>DBCC SHOW_STATISTICS ("dbo.table_name" , stat1) WITH stat_header;</code></td></tr>
<tr><td rowspan="3">6.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
</table>

*8.4. tabula – **Microsoft SQL Server** praktiskās daļas 4. soļa rezultāti*

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas EMPLOYEES tabulā, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td>2.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
98.75%</td></tr>
<tr><td rowspan="3">3.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 7.04 milisekundes, izmantojot komandu katrai tabulai HR shēmā:
UPDATE STATISTICS dbo.table_name;</td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
SELECT COUNT(*) FROM dbo.table_name;
DBCC SHOW_STATISTICS ("dbo.table_name" , stat1) WITH stat_header;</td></tr>
<tr><td rowspan="3">6.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
</table>

*8.4. tabula – **Microsoft SQL Server** praktiskās daļas 5. soļa rezultāti*

<table>
<tr><th>NPK</th><th>Darbības nosaukums</th><th colspan="2">Rezultāti</th></tr>
<tr><td>1.</td><td>Datu ievade</td><td colspan="2">Ievadītas 100 000 rindas EMPLOYEES tabulā, jo ir nepieciešams tikai uzskatāmi redzēt, kā tiek vāktas statistikas un kā strādā optimizators pie lielākām datu tabulām. Citās tabulās datus neievada.</td></tr>
<tr><td>2.</td><td>Procentuālās novirzes</td><td colspan="2">EMPLOYEES tabulas procentuālā novirze:
75.43%</td></tr>
<tr><td rowspan="3">3.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
<tr><td>4.</td><td>Statistiku vākšana</td><td colspan="2">Izpildes laiks ir 12.17 milisekundes, izmantojot komandu katrai tabulai HR shēmā:
<code>UPDATE STATISTICS dbo.table_name;</code></td></tr>
<tr><td>5.</td><td>Procentuālās novirzes</td><td colspan="2">Visas tabulas un shēmas ir ar 0% procentuālo novirzi, jo tabulu apjoms ir neliels. Izmantotā komanda:
<code>SELECT COUNT(*) FROM dbo.table_name;</code>
<code>DBCC SHOW_STATISTICS ("dbo.table_name" , stat1) WITH stat_header;</code></td></tr>
<tr><td rowspan="3">6.</td><td rowspan="3">SQL pieprasījumu izpildes plāni</td><td>1.</td><td>  </td></tr>
<tr><td>2.</td><td>  </td></tr>
<tr><td>3.</td><td>  </td></tr>
</table>

Bakalaura darbs „SQL datubāžu optimizācijas iespējas un salīdzinājums” izstrādāts LU Datorikas fakultātē.

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

Autors: Lauma Šivare

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: docents Dr.sc.comp. Aivars Niedrītis

Recenzents: docente Dr.sc.comp. Darja Sodovņikova

Darbs iesniegts Datorikas fakultātē 01.06.2015.

Dekāna pilnvarotā persona: metodiķe Ārija Sproģe

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

Komisijas sekretārs: