TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Andmebaasi optimeerimine ning selles toimuvate operatsioonide jõudluse analüüs ühe Oracle andmebaasi näitel

Bakalaureusetöö

Üliõpilane: Dan Rodionov

Üliõpilaskood: 124003 IAPB

Juhendaja: dotsent Erki Eessaar

Tallinn

2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

....

(*kuupäev*) (*allkiri*)

Annotatsioon

Rodionov D. (2015) Andmebaasi optimeerimine ning selles toimuvate operatsioonide jõudluse analüüs ühe Oracle andmebaasi näitel. Bakalaureusetöö, Tallinna Tehnikaülikool.

Käesoleva bakalaureusetöö eesmärgiks on Oracle tabelite optimeerimise meetodite uurimine ning nende mõju hindamine päringute ja andmemuudatuste kiirusele. Töö käigus luuakse sama kontseptuaalse andmemudeli põhjal kaks erinevat andmebaasi. Ühes andmebaasis on baastabelid ja neile loodud vaated, teises aga tabelid mis on loodud kasutades erinevaid andmebaasisüsteemi poolt pakutavaid optimeerimisvõimalusi ning hetktõmmiseid. Uuringu käigus tutvutakse eksperimendis kasutatavate optimeerimisvõimalustega. Eksperimendi osas aga valitakse tabelitele sobivad optimeerimisvõimalused ning koostatakse päringud ja andmemuudatused, mille põhjal andmebaasis toimuvate operatsioonide jõudlust mõõta. Seejärel täidetakse mõlema disainiga andmebaasid ühesuguste testandmetega ning mõõdetakse päringute ja andmemuudatuste täitmiskiiruseid. Samuti leitakse mõlema andmebaasi andmemahud. Töö lõpuks analüüsitakse kogutud andmeid ja võrreldakse eksperimendis kasutatud disaine operatsioonide kiiruse ja andmemahtude seisukohalt.

Bakalaureusetöö on kirjutatud eesti keeles ja koosneb 78 leheküljest. Töö sisaldab 2 joonist ning 32 tabelit.

Abstract

Rodionov D. (2015) Database Optimization and Analysis of Performance of Operations in it in the Example of an Oracle Database. Bachelor's Thesis, Tallinn University of Technology.

The goal of this thesis is to examine different optimization methods of Oracle database tables and their impact on the performance of queries and data modifications. In the process, two different databases are created, which are based on the same conceptual data model. One database consists of heap organized tables and views on top of them, while the other database uses different optimization methods of tables as well as snapshots. Optimization methods that are used in the experiment, are introduced during the research. In the experiment, appropriate optimization methods as well as queries and data modifications for analyzing the performance of query and data modification operations are selected. After that both databases are filled with identical test data and performance of queries as well as data modifications is measured. In addition, the data volumes of both databases are measured. Finally the results of experiments are analyzed and database designs are compared in terms of performance and data volumes.

Bachelor's thesis is written in Estonian and consists of 78 pages. Thesis includes 2 figures and 32 tables.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

Sissejuhatus

Infosüsteem koosneb erinevatest objektidest, millest igaühel on oma ülesanne. Objekti eesmärgiks võib näiteks olla andmete kogumine, töötlemine või salvestamine. Üheks oluliseks objektiks infosüsteemis on andmebaas, kuhu salvestatakse andmed hilisemaks töötlemiseks. Andmebaase võib olla erinevad, kuid enamlevinud on sellised andmebaasisüsteemid, mis kasutavad SQL keelt (*structured query language*). Selliseid andmebaase nimetatakse SQL-andmebaasideks.

SQL-andmebaasi kontseptuaalne kiht koosneb baastabelitest, mis on omakorda jagatud ridadeks ja veergudeks. Baastabelite struktuur võib olla erinev ning osad tabelid võivad olla omavahel seotud. Seetõttu võib sama valdkonna infosüsteemide disain olla erinev ning erineda kahe infosüsteemi vahel oluliselt.

Andmebaasi projekteerimise käigus luuakse kontseptuaalne andmemudel, mis kirjeldab ära infosüsteemi baasandmete vajadused. Selle põhjal on võimalik luua nõuetele vastav andmebaas. Andmebaasi loomiseks ei ole kindlat lahendust, kuid selle protsessi käigus tehtavad valikud mõjutavad hilisemat andmebaasi jõudlust. Seetõttu on infosüsteemi loomisel vajalik teha õigeid otsuseid, et hilisemaid probleeme vältida.

Antud töö eesmärgiks on uurida Oracle andmebaasi tabelite erinevaid optimeerimisvõimalusi ning nende mõju andmebaasi jõudlusele. Uuritavaks andmebaasisüsteemiks on valitud Oracle, kuna sellesse on sisseehitatud väga palju optimeerimisvõimalusi. Tegemist on töö kirjutamise hetke seisuga (kevad 2015) kõige populaarsema andmebaasisüsteemiga (DB-Engines Ranking) ning autoril on selle süsteemiga varasemaid kokkupuuteid seoses õppetööga ja soov oma teadmisi sellest süsteemist süvendada.

Selle saavutamiseks luuakse ühe kontseptuaalse andmemudeli alusel kaks erineva disainiga andmebaasi. Üks andmebaas kasutab tavalisi baastabeleid ja nende peale loodud vaateid, teine aga erinevaid tabelite loomiseks pakutavaid optimeerimisvõimalusi ning hetktõmmiseid. Seejärel täidetakse mõlemad andmebaasid ühesuguste testandmetega ning viiakse läbi eksperiment. Eksperimendi käigus katsetatakse mõlemas andmebaasis ühesuguseid päringuid ja andmemuudatusi, et hinnata erinevate disainide mõju andmebaasis toimuvate

operatsioonide jõudlusele. Samuti vaadeldakse baastabelite ja hetktõmmiste salvestamiseks kulunud andmemahtusid. Lõpuks jõutakse järeldusele tehtud optimeerimisotsuste mõju kohta andmebaasis toimuvate operatsioonide jõudlusele ning andmete salvestamiseks kuluvale mahule.

1. Optimeerimisvõimalused Oracle andmebaasis

Antud peatüki koostamisel on kasutatud raamatut „Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions, Second Edition“ (Kyte, 2010).

Kuna Oracle andmebaasi optimeerimiseks on mitmeid erinevaid võimalusi, siis antud töö raames tutvutakse lähemalt ainult nendega, mis on vajalikud eksperimendi läbi viimiseks.

1.1. Tabelid

Oracle andmebaasis on erinevaid baastabelite liike, mis eristuvad selle järgi, kuidas andmeid andmebaasi sisemisel tasemel esitatakse. Baastabel on tabel, mis pole defineeritud teiste tabelite põhjal. Edaspidi nimetan baastabeleid ka lihtsalt tabeliteks. Andmete sisemisel tasemel organiseerimise muutmine tähendab tänu füüsilisele andmete sõltumatusele, et andmestruktuuride muutmine ei tingi vajadust kirjutada ümber päringuid, andmemuudatusi ja rakendusi, mis neid tabeleid kasutavad.

- „Tavaline“ baastabel (*heap organized table*)
- Indeks-orienteeritud tabel (*index organized table*)
- Klastrisse koondatud tabel (*index clustered table*)
- Räsiklastrisse koondatud tabel (*hash clustered table*)
- Sorteeritud räsiklastrisse koondatud tabel (*sorted hash clustered table*)
- Välistabel (*external table*)
- Ajutine baastabel (*temporary table*)

Lisaks on võimalikud ka sellised tabelid, mis muudavad andmete esitamist andmebaasi kontseptuaalsel tasemel.

- Pesastatud tabel (*nested table*)
- Objekttabel (*object table*)

Järgnevates alampeatükkides tutvutakse põhjalikumalt indeks-orienteeritud ja räsiklastrisse koondatud tabeliga.

1.1.1. Indeks-orienteeritud tabel

Indeks-orienteeritud tabelis kasutatakse andmete hoidmiseks indeksipõhist struktuuri. Kui tavalises baastabelis salvestatakse andmed kuhja (*heap*) põhimõttel, st esimesele süsteemi leitud vabale kohale, siis indeks-orienteeritud tabelis on andmed salvestatud ja sorteeritud primaarvõtme põhjal, st igal real on oma kindel koht, kus see tuleb andmebaasi sisemisel tasemel paigutada. Väliselt käituvad indeks-orienteeritud tabelid nagu tavalised baastabelid ning nende puhul kasutatakse samasuguseid SQL lauseid.

Indeks-orienteeritud tabelis on andmed salvestatud primaarvõtme alusel loodud indeksi struktuuri. Seega pole vaja tabeli andmeid ja primaarvõtme indeksit eraldi salvestada. Kui andmeid otsida primaarvõtme väärtuse või väärtuste vahemiku järgi, siis on selles päringute täitmiskiirused paremad kui baastabelis. Sarnaste primaarvõtme väärtustega read paiknevad salvestusstruktuuris lähestikku ja neid on kiirem lugeda. Kuna primaarvõtme indeksi salvestamiseks ei ole eraldi ruumi tarvis, siis on indeks-orienteeritud tabelil baastabelist väiksemad andmemahud.

1.1.2. Räsiklastrisse koondatud tabel

Tabelite klastrisse koondamine tähendab nende tabelite andmete ühendamist andmebaasi sisemisel tasemel, kuid andmebaasi kontseptuaalsel tasemel jäävad tabelid kokku. Räsiklastrisse koondatud tabel on klastrisse koondatud tabelite alamliik. Räsiklastrisse koondatud tabelis on klastri võtmele (tabelite ühendamise aluseks olevatele veergudele) indeksiks selles sisalduvad andmed, mistõttu füüsilist indeksit ei kasutata. Andmebaasisüsteem võtab rea võtmeväärtuse, leiab sellele vastava räsiväärtuse (kas süsteemi või kasutaja poolt defineeritud funktsiooni põhjal) ning salvestab andmed plokki.

Räsiklastri suurus on määratud kohe alguses, mistõttu on oluline leida klastri loomisel sellele vastav räsivõtmete arv ning suurus. Kuigi räsivõtmete arv klastris salvestatavate andmete hulka ei mõjuta, siis määrab see ära unikaalsete võtmete arvu. Kui leitud räsivõtmete arv on liiga väike, siis võivad klastrisse andmete lisamisel tekkida kollisioonid (väärtuste kokkupõrked), mis mõjutavad andmebaasi jõudlust.

Räsiklastrit on kasulik kasutada olukordades, kus on teada kui palju ridu üks tabel hakkab tulevikus sisaldama. Nii on võimalik arvutada õige räsivõtmete arv ja klastri

suurus ning parandada andmebaasi jõudlust. Kuna räsiklastris salvestatud andmetele pääsetakse ligi räsiväärtuste põhjal, siis tuleks seda kasutada tabelites, kus on palju päringuid andmete otsimiseks.

1.2. Sektsioonideks jagamine

Sektsioonideks jagamine võimaldab hallata suuri tabeleid ja indekseid, kasutades „jaga ja valitse“ strateegiat. Andmebaasi kontseptuaalsel tasemel paistab selline tabel kasutajatele endiselt kui terviklik tabel, kuid andmebaasi sisemisel tasemel on see tabel jagatud väiksemateks osades, mida süsteemil on kergem töödelda. Tabeleid sektsioonideks jagades valitakse sektsiooni võti (veergude hulk), milles olevate andmete alusel paigutatakse andmed neile vastavatesse sektsioonidesse.

Hetkel on võimalik Oracle andmebaasis jagada tabeleid sektsioonidesse kuue meetodi põhjal:

- vahemike alusel (*range partitioning*)
- räsiväärtuste alusel (*hash partitioning*)
- nimekirja alusel (*list partitioning*)
- intervalli alusel (*interval partitioning*)
- viite alusel (*reference partitioning*)
- kombinatsiooni alusel (*composite partitioning*) (tabel jagatud sektsioonideks ühe meetodi alusel ning need sektsioonid saab omakorda jagada sektsioonideks mõne teise meetodi alusel)

Antud töös vaadatakse lähemalt vahemike, nimekirja ja viite alusel sektsioonideks jagamist.

1.2.1. Vahemike alusel sektsioonideks jagamine

Vahemike alusel tabeleid sektsioonideks jagades sisaldab iga sektsioon ainult neid ridasid, mille väärtused jäävad ette antud vahemikku. Valitud vahemikud peaksid olema pidevad, kuid mitte üksteist katma. Kui tabelisse lisatakse andmeid, mille väärtused ei lange ette antud vahemikega kokku, siis andmeid tabelisse lisada ei saa.

Heaks kandidaadiks vahemike alusel sektsioonideks jagamisel on veerud, kus hoitakse kuupäevi või ajatempleid. Kõik mingisse ajaperioodi kuuluvad read saab lasta paigutada ühte sektsiooni. Vahemike alusel sektsioonideks jagamine on kasulik, kui on vaja korraga otsida või muuta kindlas vahemikus olevaid andmeid. Näiteks, kui andmebaasis on palju päringuid, mis kasutavad oma tingimuses sektsiooni võtmeks olevat veergu, siis saab andmebaasisüsteem päringu täitmisel ebavajalikke andmeid sisaldavad sektsioonid vaatluse alt välja jätta ning päringu täitmiskiirused on selle võrra väiksemad.

1.2.2. Nimekirja alusel sektsioonideks jagamine

Nimekirja alusel sektsioonideks jagamine on sarnane vahemike alusel sektsioonideks jagamisega. Ainsaks erinevuseks on, et kui vahemike alusel sektsioonideks jagatud tabeli puhul on defineeritud väärtuste vahemikud, siis nimekirja alusel sektsioonideks jagatud tabelil saab määrata ära nimekirja väärtustest, mille alusel read ühte sektsiooni paigutatakse. Samuti nagu ka vahemike alusel sektsioonideks jagatud tabeli puhul, ei saa ka nimekirja alusel sektsioonideks jagatud tabelisse lisada andmeid, mille väärtused ei lange kokku etteantud nimekirjas olevate väärtustega.

1.2.3. Viite (välisvõtme) alusel sektsioonideks jagamine

Viite alusel sektsioonideks jagamisel kasutatakse strateegiat, kus välisvõtmega seotud tabelite korral on sõltuv tabel jagatud sektsioonideks nii, et iga sektsioon on üks-ühele vastavuses primaarse tabeli sektsiooniga.

Viite alusel sektsioonideks jagamine on kasulik olukordades, kus sõltuva tabeli andmed on seotud primaarses tabelis olevate andmetega.

1.3. Sektsioonideks jagamise indeksid

Indeksid, nagu ka tabelid, võivad olla sektsioonideks jagatud. Indeksite sektsioonideks jagamiseks on kaks võimalust:

- lokaalne indeks (*local index*)
- globaalne indeks (*global index*)

Järgnevas jaotises vaadatakse lähemalt lokaalset indeksit.

1.3.1. Lokaalne sektsioonideks jaotatud indeks

Lokaalse sektsioonideks jaotatud indeksi puhul on tabeli sektsioonide ja indeksi sektsioonide vahel 1:1 vastavus. Kõik kirjed antud indeksi sektsioonis on seotud ühe tabeli sektsiooniga ja kõik read ühes tabeli sektsioonis on vastavalt seotud ühe indeksi sektsiooniga. Globaalse sektsioonideks jaotatud indeksi puhul on tabelite ja indeksite sektsioonide vahel 1:M vastavus. See tähendab, et ühele indeksi sektsioonile võib vastata mitu tabeli sektsiooni. (Burelson, 2010)

Lokaalsed indeksid jaotuvad kaheks:

- lokaalsed eesliitega indeksid (*local prefixed index*) – indeksid, kus sektsioonide võtmed on indeksisse kuuluvate veergude hulgas kõige esimesed.
- lokaalsed eesliiteta indeksid (*local nonprefixed index*) – indeksid, kus sektsioonide võtmed ei ole indeksisse kuuluvate veergude hulgas kõige esimesed. Sellisel juhul indeksid võivad, aga ei pruugi, sektsioonide võtmeid sisaldada

Lokaalsete eesliitega indeksite korral on garanteeritud, et kui päringu tingimuses viidatakse sektsiooni võtmeks olevatele veergudele, siis andmebaasisüsteem saab kasutada indeksite sektsioonide välistamist, st päringu täitmiseks loeb andmebaasisüsteem osasid, aga mitte kõiki indeksi sektsioone. Lokaalsete eesliiteta indeksite puhul pole see garanteeritud. Seetõttu võib tekkida olukordi, kus päringud on kiiremad siis kui on kasutusel eesliitega indeksid, võrreldes olukorraga, kui kasutusel on eesliiteta indeksid.

Üldjuhul on lokaalsed indeksid globaalsetest indeksitest stabiilsemad, kuna iga indeksi sektsioon on seotud ainult ühe tabeli sektsiooniga. Globaalse indeksi puhul viitab aga üks indeksi sektsioon mitmele tabeli sektsioonile, mistõttu probleemide tekkimisel indeksis on takistatud mitmete tabeli sektsioonide kasutamine selle indeksi vahendusel. Üksiku tabeli sektsiooni kustutamine mõjutab näiteks kogu globaalset sektsioonideks jagatud indeksit.

1.4. Hetktõmmis

Hetktõmmis e materialiseeritud vaade (*materialized view*) on tuletatud (teiste tabelite põhjal defineeritud) tabel, milles olevaid andmeid uuendatakse päringu põhjal perioodiliselt ning soovi korral ka erakorraliselt. Neist võib mõelda kui ette valmis arvutatud ja eraldi koopiana salvestatud päringutulemustest. Tabeli uuendamiseks päritakse andmed lokaalsetest või välistest tabelitest, mille põhjal hetktõmmis loodi. Selleks, et hetktõmmiseid kiiresti luua ja värskendada, on vaja et vajalikel tabelitel oleks loodud nendes toimunud andmemuudatuste jälgimiseks mõeldud logid.

Hetktõmmiseid saab luua koheselt (*immediate*) või esimesel nõutud uuendamisel (*deferred*).

Hetktõmmise uuendamiseks on aga kolm viisi:

- kiiresti (*fast*) – hetktõmmist üritatakse uuendada kiiresti. Kui hetktõmmise aluseks olevatel baastabelitel logid puuduvad, siis uuendamine ebaõnnestub.
- täielikult (*complete*) – hetktõmmist sisaldav tabeli segment tühjendatakse ja seejärel täidetakse uuesti päringust saadud andmetega.
- sunniviisiliselt (*force*) – hetktõmmist üritatakse uuendada kiiresti. Kui see ei õnnestu, uuendatakse hetktõmmist täielikult.

Hetktõmmise uuendamist saab esile kutsuda kas andmemuudatust sisaldanud transaktsiooni kinnitamisel (*on commit*) või selle nõudmisel (*on demand*) (Materialized Views in Oracle).

Hetktõmmiseid on näiteks kasulik kasutada olukordades, kus on vaja vähendada andmebaasi võrgukoormust. Sel juhul on võimalik andmebaas jagada väiksemateks osadeks, et andmete pärimine toimuks mitmest andmebaasi serverist ning nendes serverites on lugemiseks mõeldud hetktõmmised. Samuti võib olla hetktõmmiseid kasulik kasutada siis, kui andmebaasis on keerukad päringud suurte andmehulkade põhjal ning nende päringute täitmine otse baastabelite põhjal võtab palju aega. Kui sellise päringu tulemus eelnevalt välja arvutada, siis saab kasutajatele kiiresti päringu vastuse anda. Mündi teiseks küljeks on muidugi see, et kui hetktõmmise aluseks olevates baastabelites andmed muutuvad, siis tuleb muuta ka hetktõmmist. Kui seda teha sünkroonselt, siis võtavad andmemuudatused rohkem aega, kuid hetktõmmises

on kõige värskemad andmed. Kui seda teha asünkroonselt, siis võtavad andmemuudatused vähem aega, kuid hetktõmmistes pole enam suurel osal ajast kõige värskemad andmed. (Materialized View Concepts and Architecture).

2. Eksperimendi andmebaasi projekteerimine

Eksperimendis kasutatavaks valdkonnaks on valitud arvutitoodete internetipood. Organisatsiooni eesmärgiks on pakkuda laia valikut IT-tooteid kiirelt ning mugavalt. Kuna loodud andmebaasi on eksperimendi huvides lihtsustatud, siis ei ole see mõeldud reaalseks kasutamiseks. Samas võib aga antud disaini kasutada reaalse süsteemi väljaarendamisel. Andmebaaside projekteerimisel on võetud aluseks autori poolt õppeainetes „Andmebaasid I“ (Sams et al., 2014) ja „Andmebaasid II“ (Sams ja Rodionov, 2014) koos kaaslastega loodud iseseisva töö projektid.

2.1. Terviksüsteemi ülevaade

Järgnevalt esitatakse ülevaade arvutitoodete internetipoe toimimisest. Antud töös vaadeldakse täpsemalt **tellimuste funktsionaalset allsüsteemi**.

2.1.1. Lausendid

- Administraator lisab toote infosüsteemi.
- Administraator redigeerib tooteinfot.
- Administraator arhiveerib toote.
- Administraator eemaldab toote infosüsteemist.
- Tooteinfo sisaldab näitajaid.
- Pood sisaldab tooteid.
- Klient külastab poodi.
- Klient ostab tooteid.
- Müüja müüb tooteid.
- Müümisel esitatakse arve.
- Kasutajal on konto.
- Konto sisaldab informatsiooni kasutaja kohta.
- Kasutaja otsib tooteid.
- Kasutaja filtreerib otsingu.
- Kontoga lisatakse toode tellimusse.
- Kontoga toimub tellimuse esitamine.
- Tellimuse esitamisel esitakse kasutajale arve.
- Kontoga jälgitakse tellimuse seisu.

- Müüja kontrollib tellimust.
- Müüja uuendab tellimuse seisu: esitatud, kinnitatud, tühistatud, tellimisel, kohal, saadetud.
- Tellimuse täitmisel vähendatakse tootekogust.
- Poodide tootekogused on ühised.

2.1.2. Põhiobjektid

- Toode
- Tellimus
- Kasutaja
- Töötaja
- Klassifikaator
- Tarnija
- Tarne
- Tootja

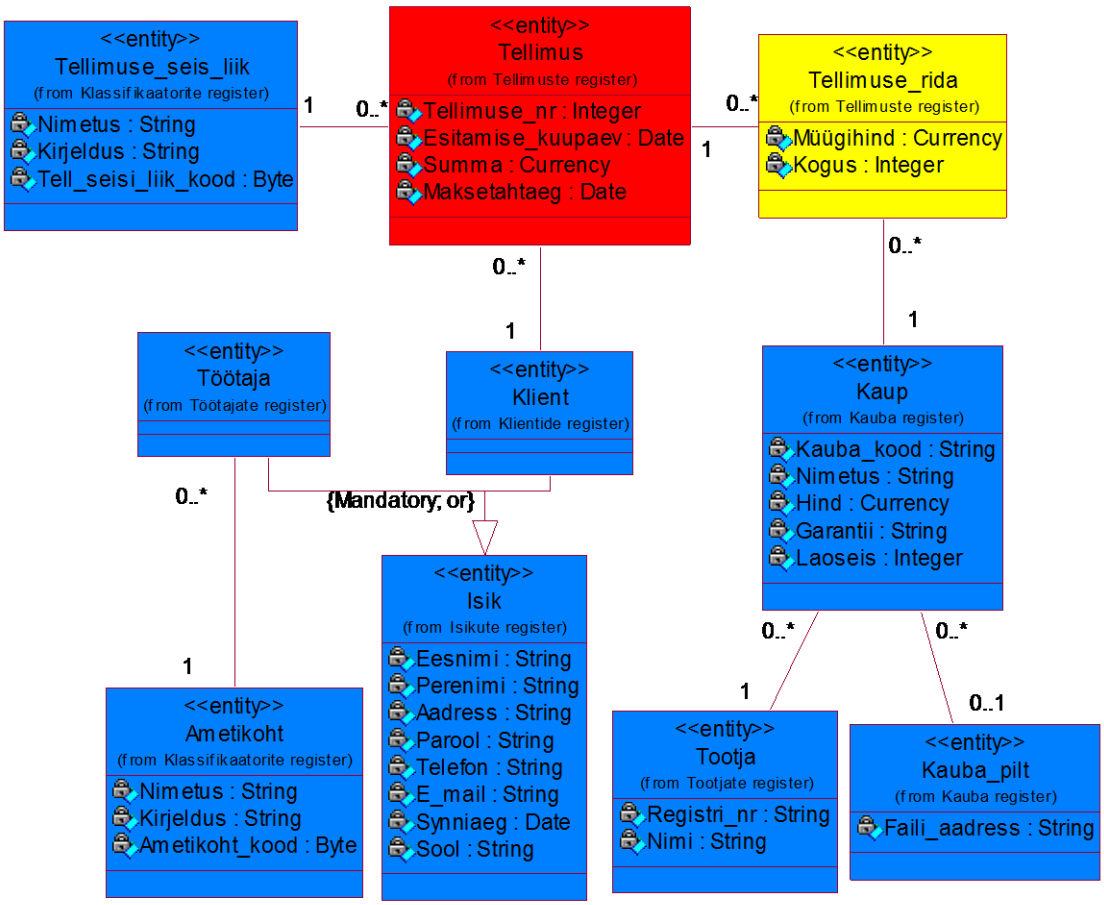
2.1.3. Põhiprotsessid

- Toote lisamine.
- Tooteinfo muutmine.
- Toote eemaldamine.
- Toote otsimine.
- Tooteinfo pärimine.
- Tellimuse haldamine.
- Tellimuse seisu muutmine.
- Arve esitamine.

2.2. Kontseptuaalne andmemudel

Kontseptuaalne andmemudel on esitatud olemi-suhte diagrammina (vt joonis 1). Diagrammi loomiseks kasutati CASE vahendit Rational Rose Enterprise Edition. Diagrammil esitatud olemitüübid ja nende definitsioonid on ära toodud tabelis 1. Olemitüüpide atribuutide definitsioonid on ära toodud tabelis 2.

2.2.1. Olemi-suhte diagramm



Joonis 1. Tellimuste registri olemi-suhte diagramm

2.2.2. Olemitüüpide definitsioonid

Tabel 1. Olemitüüpide definitsioonid

Olemitüübi nimi (aliased)	Kuuluvus registrisse	Definitsioon
Kaup	Kaupade register	Kaup on üksik ese poes müüdavate toodete seast.
Isik	Isikute register	Isik on töötaja ja kliendi üldistus.
Töötaja	Töötajate register	Töötaja on poe toimimisega seotud isik.
Klient	Klientide register	Klient on poodi külastav registreerunud isik.
Kauba pilt	Kaupade register	Kauba pilt on aadress, mis viitab kaubaga seotud pildile.
Tootja	Tootjate register	Tootja on kauba valmistaja.

Olemitüübi nimi (aliased)	Kuuluvus registrisse	Definitsioon
Tellimuse rida	**Tellimuste register**	Tellimuse rida viib kokku tellimuse ja selles sisalduvad tooted.
Tellimus	**Tellimuste register**	Tellimus on kliendi poolt esitatud dokument, mille põhjal töötaja väljastab toote(d) ning esitab arve.
Tellimuse seisundi liik	Klassifikaatorite register	Tellimuse seisundi liik on tellimuse omadus, mille põhjal saab töötaja hinnata selle seisu (esitatud, kinnitatud, tühistatud, tellimisel, kohal, saadetud). Seisundi üldistus on klassifikaator.
Ametikoht	Klassifikaatorite register	Ametikoht on töötaja amet ettevõttes. Ametikoha üldistus on klassifikaator.

2.2.3. Atribuutide definitsioonid

Tabel 2. Atribuutide definitsioonid

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon	Näiteväärtus
Kaup	kauba_kood	Kaupa unikaalselt identifitseeriv kood, mida kasutatakse kaubale viitamiseks ka väljaspool andmebaasi.	TJ0HS7VCYF
	nimetus	Kaupa identifitseeriv nimi, mida kasutatakse kaubale viitamiseks ka väljaspool andmebaasi.	Topdankix
	hind	Kauba müügihind eurodes. Hind sisaldab käibemaksu.	99,07
	garantii	Kauba garantii loetava tekstina, mida esitatakse kauba juures.	2 aastat
	laoseis	Kauba laoseis poes. Laoseisu ühikuks on tükk (tk).	4
Isik	isik_id	Isikut unikaalselt identifitseeriv täisarv.	1
	eesnimi	Isikut identifitseeriv eesnimi.	Anna
	perenimi	Isikut identifitseeriv perekonnanimi.	Tucker
	synniaeg	Isiku sünniaeg kuupäeva täpsusega.	09.01.1948
	aadress	Isiku elukoht.	8 Corben Point

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon	Näiteväärtus
	e_mail	Isikut unikaalselt identifitseeriv meiliaadress, mida kasutatakse ka kasutajanimena. Meiliaadress on vajalik isikute vaheliseks suhtluseks.	atucker0@freewebs.com
	parool	Isiku parool, mida kasutatakse kasutaja verifitseerimiseks.	Atucker0
	telefon	Kliendi telefoninumber. Telefoninumbrit kasutatakse kliendi ja töötaja vaheliseks suhtluseks.	43796769096
Tellimus	tellimuse_nr	Tellimuse unikaalne identifitseeriv täisarv, mida kasutatakse tellimusele viitamiseks ka väljaspool andmebaasi.	1
	summa	Tellimuses sisalduvate toodete maksumus eurodes. Summa moodustatakse tellimuses olevate toodete ning nende koguste korrutiste summana.	724,49
	esitamise_kuupaev	Kuupäev, millal tellimus süsteemile edastati	9.12.2013
	makse-tahtaeg	Kuupäev, milleks peab tellimus kinnitatud olema. Tellimuse mitte kinnitamise korral see tühistatakse.	23.12.2013
Tootja	registri_nr	Tootja unikaalne identifitseeriv kood, mida kasutatakse tootjale viitamiseks ka väljaspool andmebaasi.	9N799528
	nimi	Tootja firma nimetus.	Linklinks
Tellimuse_rida	myygihind	Kauba tellimuse esitamisel määratud hind eurodes. Kauba müügihind sisaldab käibemaksu.	62,47
	kogus	Kogus mis näitab, mitu ühikut mingit kaupa antud tellimuses sisaldub. Koguse ühikuks on tükk (tk).	5

3. Eksperimendi kirjeldus

Eksperimendis kasutatakse Oracle Database 12c Enterprise Edition Release 1 andmebaasisüsteemi ning andmebaasidega töötamiseks kasutatakse Oracle SQL Developer tarkvara versiooni 4.0.3.16. Eksperiment tehakse ülikooli poolt pakutavas serveris apex.ttu.ee, millel on järgmised tehnilised omadused: 40 GB suvapöördusmälu (*RAM*), 15 keskprotsessorit (*CPU*) ja CentOS 6.4 operatsioonisüsteem.

3.1. Eksperimendis kasutatava andmebaasi loomine

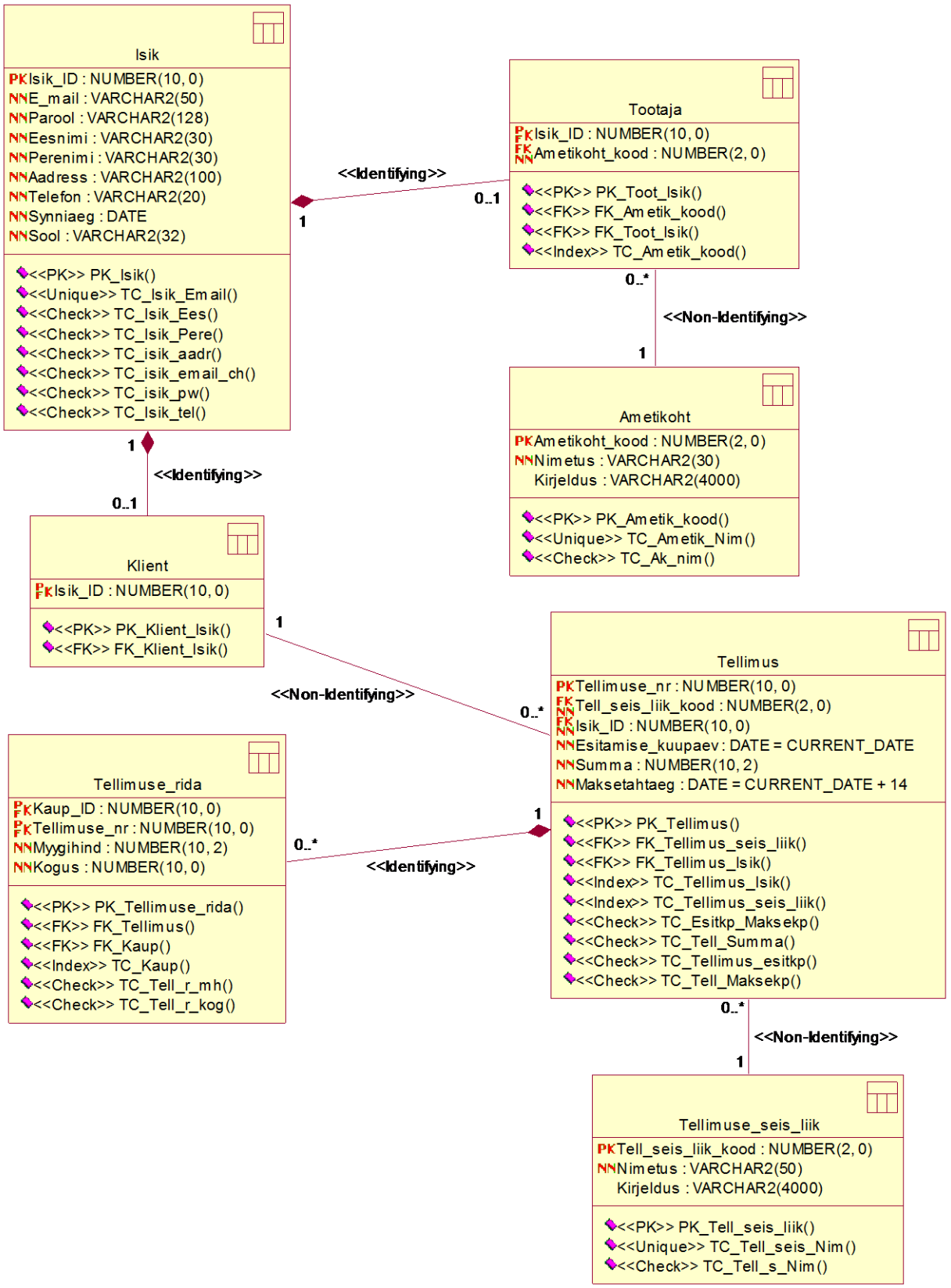
Katse käigus luuakse andmebaasi kontseptuaalse kirjelduse (vt joonis 1) alusel kaks erineva füüsilise disainiga andmebaasi (loomise koodi vt lisa 1 ja lisa 2), millest üks andmebaas kasutab „tavalisi“ tabeleid ning teine optimeeritud variante nendest tabelitest. Nimetan neid disaine edaspidi vastavalt „tavaline“ ja „optimeeritud“. Andmebaaside loomisel on võetud aluseks autori poolt õppeainetes „Andmebaasid I“ (Sams et al., 2014) ja „Andmebaasid II“ (Sams ja Rodionov, 2014) koos kaaslastega loodud iseseisva töö projektid.

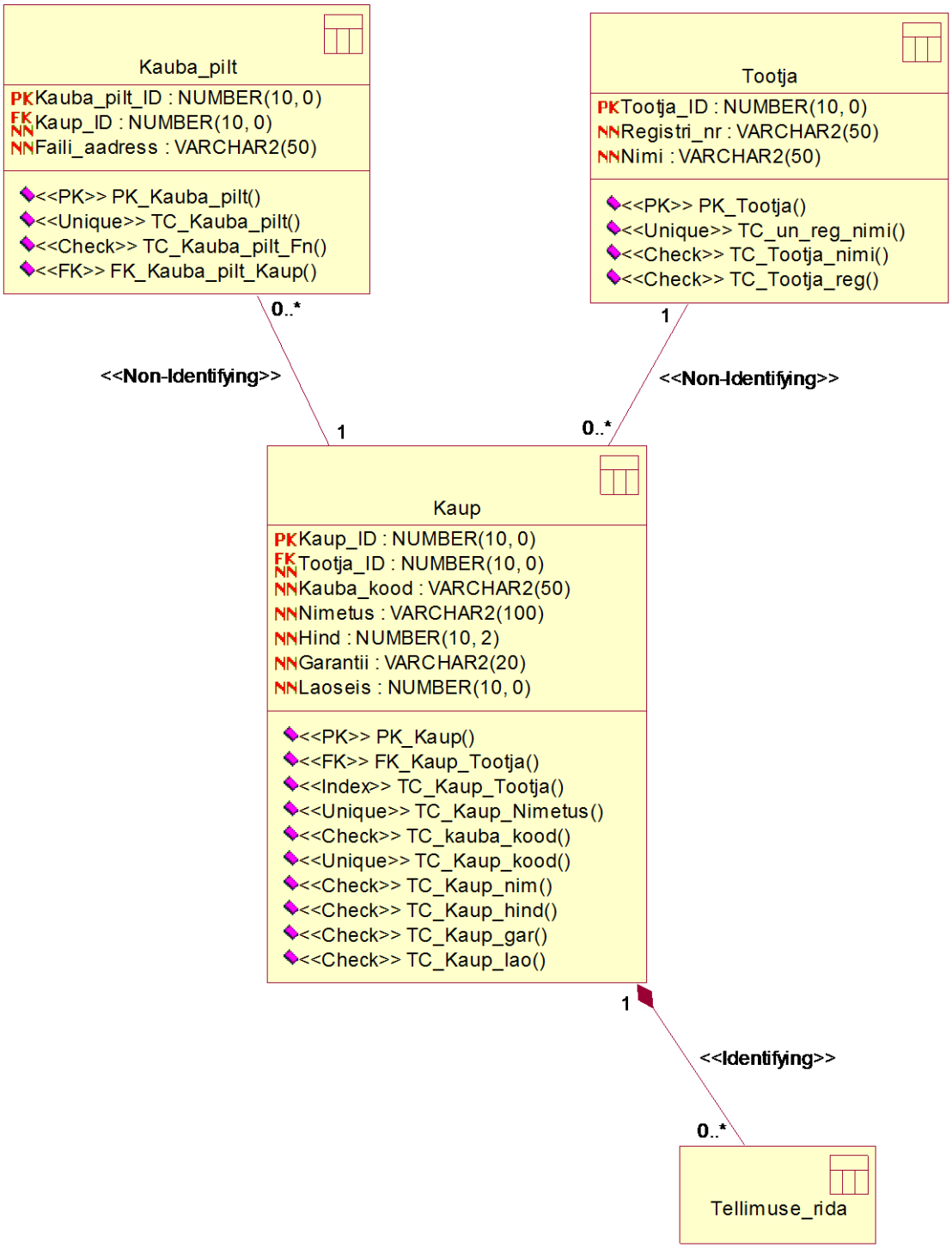
Lisaks kontseptuaalsest mudelist tulenevatele baastabelitele luuakse süsteemis ka järgmised vaated.

- *Tell_Nimek* – esitatakse tellimuse esitaja ees- ja perenimi, email, tellimuse üldinformatsioon ning tellimuse seisund (vt lisa 3).
- *Tell_Detail* – esitatakse kauba nimi, informatsioon ning kogus tellimuses (vt lisa 4).

Tavalise disaini korral luuakse baastabelid, B-puu indeksid ja vaated. Optimeeritud disaini korral kasutatakse erinevaid Oracle jõudluse parandamiseks mõeldud lisavõimalusi nagu indeks-orienteeritud tabeleid, tabelite sektsioonideks jagamist, andmete pakkimist, klastreid ja hetktõmmiseid e materialiseeritud vaateid („tavaliste“ vaadete asemel). Sektsioonideks jagatud tabelite välisvõtmetele luuakse üldjuhul globaalsed sektsioonideks jagamata indeksid. Ühel juhul luuakse lokaalne sektsioonideks jagatud indeks. Kasutajate (programmide) jaoks on nii tavalise disaini kui optimeeritud disaini korral baastabelite struktuur ühesugune ja seda kirjeldab joonis 2. Erinevus on selles, kuidas nendes tabelites olevaid andmeid

andmebaasisüsteemis sisemiselt salvestatakse. Ühesugune on ka vaadete struktuur. Erinevus on selles, kas vaate väärtus arvutatakse jooksult kui kasutaja vaate poole pöördub („tavalised vaated“) või on vaadete väärtused ette valmis arvutatud („hetktõmmised“).


Isik
PK Isik_ID : NUMBER(10, 0)
NN E_mail : VARCHAR2(50)
NN Parool : VARCHAR2(128)
NN Eesnimi : VARCHAR2(30)
NN Perenimi : VARCHAR2(30)
NN Aadress : VARCHAR2(100)
NN Telefon : VARCHAR2(20)
NN Synniaeg : DATE
NN Sool : VARCHAR2(32)
<<PK>> PK_Isik()
<<Unique>> TC_Isik_Email()
<<Check>> TC_Isik_Ees()
<<Check>> TC_Isik_Pere()
<<Check>> TC_isik_aadr()
<<Check>> TC_isik_email_ch()
<<Check>> TC_isik_pw()
<<Check>> TC_Isik_tel()
<<Identifying>>
1
0..1
Tootaja
PK FK NN Isik_ID : NUMBER(10, 0)
FK NN Ametikoht_kood : NUMBER(2, 0)
<<PK>> PK_Toot_Isik()
<<FK>> FK_Ametik_kood()
<<FK>> FK_Toot_Isik()
<<Index>> TC_Ametik_kood()
0..*
<<Non-Identifying>>
1
Ametikoht
PK Ametikoht_kood : NUMBER(2, 0)
NN Nimetus : VARCHAR2(30)
Kirjeldus : VARCHAR2(4000)
<<PK>> PK_Ametik_kood()
<<Unique>> TC_Ametik_Nim()
<<Check>> TC_Ak_nim()
1
<<Identifying>>
0..1
Klient
PK FK Isik_ID : NUMBER(10, 0)
<<PK>> PK_Klient_Isik()
<<FK>> FK_Klient_Isik()
1
<<Non-Identifying>>
0..*
Tellimus
PK Tellimuse_nr : NUMBER(10, 0)
FK NN Tell_seis_liik_kood : NUMBER(2, 0)
FK NN Isik_ID : NUMBER(10, 0)
NN Esitamise_kuupaev : DATE = CURRENT_DATE
NN Summa : NUMBER(10, 2)
NN Maksetahtaeg : DATE = CURRENT_DATE + 14
<<PK>> PK_Tellimus()
<<FK>> FK_Tellimus_seis_liik()
<<FK>> FK_Tellimus_Isik()
<<Index>> TC_Tellimus_Isik()
<<Index>> TC_Tellimus_seis_liik()
<<Check>> TC_Esitkp_Maksekp()
<<Check>> TC_Tell_Summa()
<<Check>> TC_Tellimus_esitkp()
<<Check>> TC_Tell_Maksekp()
Tellimuse_rida
PK FK Kaup_ID : NUMBER(10, 0)
PK FK Tellimuse_nr : NUMBER(10, 0)
NN Myygihind : NUMBER(10, 2)
NN Kogus : NUMBER(10, 0)
<<PK>> PK_Tellimuse_rida()
<<FK>> FK_Tellimus()
<<FK>> FK_Kaup()
<<Index>> TC_Kaup()
<<Check>> TC_Tell_r_mh()
<<Check>> TC_Tell_r_kog()
0..*
<<Identifying>>
1
0..*
<<Non-Identifying>>
1
Tellimuse_seis_liik
PK Tell_seis_liik_kood : NUMBER(2, 0)
NN Nimetus : VARCHAR2(50)
Kirjeldus : VARCHAR2(4000)
<<PK>> PK_Tell_seis_liik()
<<Unique>> TC_Tell_seis_Nim()
<<Check>> TC_Tell_s_Nim()



Joonis 2. Tellimuste registri loogilise disaini andmebaasi diagramm (UML notatsioonis)

apex.ttu.ee serveris paiknevas Oracle andmebaasis pannakse mõlema disaini alusel loodud skeemiobjektid samasse skeemi (*C##TEST*). Nimekonfliktide vältimiseks kasutatakse tavalise disaini objektide puhul nimedes eesliidet „TAV_“ ja optimeeritud disaini korral eesliidet „OPT_“. Nimekonfliktide vältimiseks kasutatakse neid eesliiteid ka kitsenduste nimedes.

Optimeeritud disainis kasutatavate võtete valimisele aitas kaasa nendega eelnev tutvumine (vt peatükk 1). Optimeeritud disainis on võrreldes tavalise disainiga tehtud järgmised muudatused.

- *Tellimus* tabel on jagatud sektsioonideks seisundite alusel (*list partitioning*) ning lõppseisundites olevate tellimuste andmed on pakitud (*compress*). Pakitakse seisundis „tühistatud“ või „saadetud“ (seisundi liigi koodidega 4 või 5) olevad tellimused. Tegemist on tellimuse võimalike lõppseisunditega. Lõppseisundis olevate tellimuste andmeid on vaja lugeda, kuid mitte enam muuta. Seega pakkimisest tulenev andmemuudatuste aeglasemaks muutumine ei tohiks hakata suurt rolli mängima.
- Tabel *Tellimuse_rida* on loodud indeks-orienteeritud tabelina (*index-organized table*). Kuna tabelis on veergude *Tellimuse_nr* ja *Kaup_ID* põhjal (just selles järjekorras) loodud kombineeritud võti, siis on süsteemis sisemiselt tellimuste read sorteeritud tellimuse numbri ja selle sees kauba identifikaatori järgi. See on kasulik, kui otsitakse andmeid konkreetse tellimuse või tellimuste (numbri) vahemiku järgi.
- *Kaup* tabel on jagatud sektsioonideks hinnavahemike alusel (*range partitioning*). Hinnavahemikud on järgmised 0-25, 25-50, 50-75 ning 75 ja rohkem. Ühtegi sektsiooni ei pakita.
- *Kauba_pilt* tabel on jaotatud sektsioonideks vastavalt tabeli *Kaup* sektsioonidele (*reference partitioning*). Kaupa on sageli vaja lugeda koos tema pildiga ning kaubaga seotud piltide lugemisel on süsteemil vaja vähem andmeid läbi vaadata. Tabelis olevad välisvõtme veerud indekseeritakse lokaalse sektsioonideks jagatud indeksiga (*local partitioned index*).
- Vaade *Tell_Nimek* on loodud hetktõmmisena (*materialized view*) (vt lisa 5). Nii vaated kui hetktõmmised on tabelid. Erinevus on lihtsalt selles, millal leiab süsteem nendes tabelites olevad andmed. Vaadete korral leitakse need siis, kui tehakse päring vaate põhjal kuid hetktõmmise korral on see väärtus

eelnevalt välja arvutatud ja salvestatud. *Tell_Nimek* puhul toimub värskendamine sünkroonselt (*on commit*). *Tell_Nimek* puhul toimub osaline värskendamine (*refresh fast*). Osaliseks värskendamiseks on vaja luua hetktõmmises viidatud baastabelitele logid, et süsteem saaks registreerida andmeid muudetud ridade kohta ja selle alusel hetktõmmist osaliselt värskendada (mitte nullist uuesti üles ehitada).

- Vaade *Tell_Detail* on loodud hetktõmmisena (vt lisa 6). *Tell_Detail* puhul toimub värskendamine asünkroonselt (*on demand*). *Tell_Detail* puhul toimub täielik värskendamine (*refresh complete*).
- *Tellimuse_seisundi_liik* tabeli puhul on kasutatud ühe tabeli räsiklastrit (*single table hash cluster*), et kiirendada andmete lugemist.
- *Ametikoht* ja *Töötaja* tabel on koondatud sisemisel tasemel räsiklastrisse (*hash cluster*). Need tabelid on koondatud klastrisse, sest neid kasutatakse sageli päringutes koos.

Selleks, et hinnata tabelite *Ametikoht* ja *Töötaja* põhjal loodud räsiklastri suurust, viiakse läbi eksperiment (Bad Hash Clusters). Mõlemasse tabelisse sisestatakse testandmed, värskendatakse nende tabelite statistika ning käivitatakse järgnev SQL lause.

```
ANALYZE CLUSTER OPT_Tootaja_Klaster VALIDATE STRUCTURE
CASCADE;
```

Seejärel käivitatakse järgnevad päringud, mis on esitatud koos nendele vastavate tulemustega.

```
SELECT cluster_name, cluster_type, key_size, hashkeys,
avg_blocks_per_key
FROM user_clusters;
```

Selle päringu tulemus on esitatud tabelis 3 ning see kirjeldab esialgselt plaanitud klastri suurust. Veergudes *cluster_name* ja *cluster_type* kuvatakse vastavalt klastri nimi ja tüüp. Veerus *key_size* näidatakse loodud klastri suurust baitides. Veerus *hashkeys* kuvatakse räsivõtmete arv klastris ning veerus *avg_blocks_per_key* näidatakse, mitu plokki on ühe klastri võtme kohta.

Tabel 3. Klastri Töötaja_klaster esialgne plaanitud suurus

cluster_name	cluster_type	key_size	hashkeys	avg_blocks_per_key
OPT_Tootaja_klaster	hash	64	3	1

```
SELECT count(*) as keycount, avg(c) as mean, stddev(c) as
stddev
FROM   (
       SELECT count(*) as c
       FROM OPT_Tootaja
       GROUP BY Ametikoht_kood
       );
```

Selle päringu tulemus on esitatud tabelis 4 ning see kirjeldab tabeli *Töötaja* võtmete arvu. Veerus *keycount* kuvatakse erinevate võtmete arv klastris. Veerg *mean* näitab keskmist ridade arvu ühe võtme kohta ning veerg *stddev* ridade arvu standardhälvet ühe võtme kohta. Sarnane veergude kirjeldus kehtib ka tabeli 5 korral, kus esitatakse tabeli *Ametikoht* võtmete arv.

Tabel 4. Tabeli Töötaja võtmete arv

keycount	mean	stddev
3	3,33	3,21

```
SELECT count(*) as keycount, avg(c) as mean, stddev(c) as
stddev
FROM   (
       SELECT count(*) as c
       FROM OPT_Ametikoht
       GROUP BY Ametikoht_kood
       );
```

Tabel 5. Tabeli Ametikoht võtmete arv

keycount	mean	stddev
3	1	0

```
SELECT table_name, avg_row_len
FROM user_tables;
```

Selle päringu tulemus on esitatud tabelis 6 ning see kirjeldab tabelite *Töötaja* ja *Ametikoht* keskmiseid ridade pikkuseid. Veerus *table_name* esitatakse tabeli nimi ning veerus *avg_row_len* keskmine rea pikkus selles tabelis.

Tabel 6. Tabelite Töötaja ja Ametikoht keskmine rea pikkus baitides

table_name	avg_row_len
OPT_Tootaja	6
OPT_Ametikoht	41

Saadud tulemuste põhjal on näha, et loodud räsiklaster on suurusega 64 baiti ning sisaldab 3 räsiväärtust. Tabeli *Töötaja* ridade arvuks on 10 ning tabeli *Ametikoht* ridade arvuks 3. Tabelite keskmised rea suurused on vastavalt 6 ja 41 baiti.

Kui antud andmeid analüüsida, siis selgub, et loodud räsiklaster on liiga väike. Kuna töötaja andmete keskmine suurus on 6 baiti ning ametikoha klassifikaatori andmete keskmine suurus on 41 baiti, siis näiteks seitsme müüja (müüjate arv tabelis *Töötaja*) andmete salvestamiseks koos vastava ametikoha andmetega kuluks 41+6*7=83 baiti. Seega on võtmeväärtusele vastavate andmete suurusega 64 baiti loodud räsiklaster liiga väike ning seda tuleks suurendada. Kuna ridade arv võib töö käigus muutuda, siis tuleks sellega arvestada ka klastri suuruse määramisel. Seetõttu suurendatakse esialgset ühe klastri võtmega seotud andmete suurust kahekordselt, esialgselt 64 baidilt 128 baidini. Kuna andmete hulk pole väga suur, siis pole ka põhjust muretseda liigse salvestusruumi raiskamise pärast, kuid teisalt tähendab andmete väike hulk ilmselt, et tabelite klastrisse koondamine ei anna töökiiruse mõttes suurt võitu. Klastri lõplik suurus on esitatud tabelis 7.

Tabel 7. Klastri Töötaja_klaster lõplik suurus

cluster_name	cluster_type	key_size	hashkeys	avg_blocks_per_key
OPT_Tootaja_klaster	hash	128	3	1

3.2. Eksperimendi käigus katsetatavad operatsioonid

Eksperimendi jooksul käivitatakse andmebaasisüsteemis erinevaid SQL lauseid ja mõõdetakse nende täitmiseks kuluvat aega. Laused käivitatakse nii tavalise disainiga tabelite kui ka optimeeritud disainiga tabelite põhjal, et täitmise aega oleks võimalik võrrelda. Lausete täitmise kiiruse mõõtmiseks kasutatakse TIMING funktsionaalsust, mille kasutamiseks vajalikud laused on esitatud järgnevalt.

```
SET AUTOTRACE ON
SET TIMING ON
Uuritav SQL lause;
SET TIMING OFF
SET AUTOTRACE OFF
```

Päringute töökiiruse mõõtmisel käivitatakse lause kolm korda ja esitatakse nende mõõtmiste aritmeetiline keskmine. Andmemuudatuse korral tehakse muudatus üks kord ja mõõdetakse kiirust. Peale andmemuudatust antakse muudatuse kinnitamiseks COMMIT käsk ja mõõdetakse nii lause täitmiseks kui tulemuse kinnitamiseks kulunud aega.

Füüsiline andmete sõltumatus tähendab, et muutes andmete sisemisi salvestusviise ei ole vaja muuta tabelite põhjal käivitatavaid andmebaasikeele lauseid. Seega on tavalise ja optimeeritud disaini puhul laused ühesugused (erinevused vaid tabelite nimedes).

Kui tavalise ja optimeeritud andmebaasi lausete täitmiskiirustes esineb suuri erinevusi, siis uuritakse lausete täitmisplaane, et jõuda kiiruse erinevuste põhjuste jälile. Ühtlasi võrreldakse ka mõlema disaini korral tekkivaid andmemahtusid.

Järgnevalt esitatakse katses kasutatavad päringute ja andmemuudatuste kirjeldused ning neid realiseerivad SQL laused. Iga lause korral kommenteeritakse ka selle valiku põhjust.

- Ühe kaubaga seotud piltide aadresside leidmine *Kauba_pilt* tabelist.
  ```
  SELECT Faili_aadress FROM Kauba_pilt WHERE Kaup_ID = 500;
  ```
  Valiku põhjus: Huvipakkuva faili aadressi leidmiseks saab kasutada (väiksemat) lokaalset sektsioonideks jagatud indeksit ja seega võiks optimeeritud disaini korral olla päringu täitmine kiirem kui tavalise disaini korral.

- Odavate kaupadega seotud piltide arvu leidmine.

```
SELECT K.kaup_id, Count(*) cnt FROM Kaup K
INNER JOIN Kauba_pilt KP ON K.kaup_id=KP.kaup_id
WHERE K.hind<25 GROUP BY K.kaup_id;
```

  Valiku põhjus: On vaja kokku ühendada kaks tabelit, mõlemast tabelist lugeda andmeid vaid ühest sektsioonist. Päringu täitmiseks peaks süsteemil piisama *Kauba_pilt* tabeli korral veerule *kaup_id* loodud indeksi lugemisest, tabelit ennast pole vaja puutuda. Kuna *Kauba_pilt* tabelis on (kaup_id) välisvõtmele loodud lokaalne (väikesemahulisem), sektsioonideks jaotatud indeks, siis see peaks päringut kiirendama.

- Kõikide „esitatud“ seisundis tellimuste leidmine *Tellimus* tabelist.

```
SELECT * FROM Tellimus WHERE Tell_seis_liik_kood = 4;
```

  Valiku põhjus: Andmeid tuleb lugeda ainult ühest sektsioonist ja seega võiks optimeeritud disaini korral olla päringu täitmine kiirem kui tavalise disaini korral. Lisaks on optimeeritud disaini korral andmed pakitud, mis tähendab, et loetavate plokkide hulk on väiksem ja lugemine peaks olema kiirem kui tavalise disaini korral.

- Mitme tellimusega seotud kaupade hinna leidmine *Tellimuse_rida* tabelist.

```
SELECT SUM(Myygihind) FROM Tellimuse_rida
WHERE Tellimuse_nr BETWEEN 24999 AND 25001;
```

  Valiku põhjus: Optimeeritud disaini korral on tänu indeks-organiseeritud tabelile lähestikku paiknevate numbritega tellimuste read salvestatud kettale füüsiliselt lähestikku ning tabeli andmed (sh *myygihind*) on selles samas indeksi struktuuris ja seega võiks päringu täitmine olla kiirem kui tavalise disaini korral.

- Kõikide töötajate ja nende ametikohtade leidmine.

```
SELECT T.Isik_ID, I.Eesnimi, I.Perenimi, A.Nimetus Amet
FROM (Tootaja T INNER JOIN Isik I ON T.Isik_ID=I.Isik_ID)
INNER JOIN Ametikoht A ON
T.Ametikoht_kood=A.Ametikoht_kood;
```

  Valiku põhjus: Kuna optimeeritud disaini korral on töötajate ja ametite andmed samas klastris (sisemisel tasemel ühendatud), siis võiks päringu täitmine olla kiirem kui tavalise disaini korral.

- Ühes hinnavahemikus olevate kaupade arvu leidmine.

```
SELECT COUNT(*) Arv FROM Kaup
WHERE Hind BETWEEN 1 AND 35;
```

  Valiku põhjus: Andmeid tuleb lugeda ainult kahest sektsioonist ja seega võiks optimeeritud disaini korral olla päringu täitmine kiirem kui tavalise disaini korral.

- Ühes tellimuses sisalduvate toodete nimekirja kuvamine vaate või hetktõmmise *Tell_detail* põhjal.

```
SELECT * FROM Tell_detail WHERE Tellimuse_nr=50000;
```

  Valiku põhjus: Hetktõmmise korral on väärtused eelnevalt välja arvutatud ja salvestatud, mistõttu võiks päringu täitmine olla kiirem hetktõmmise kui vaate korral.

- Kõigi tellimuste numbrite ja seisundite nimede leidmine.

```
SELECT Tellimuse_nr, Nimetus AS Seisund FROM Tellimus T
INNER JOIN Tellimuse_seis_liik TSL ON
TSL.Tell_seis_liik_kood=T.Tell_seis_liik_kood;
```

  Valiku põhjus: Pakub huvi, kuidas mõjutab ühe tabeli räsiklaster päringu kiirust.

- Ühe tellimuse lisamine.

```
INSERT INTO Tellimus (Tell_seis_liik_kood, Isik_ID,
Esitamise_kuupaev, Summa, Maksetahtaeg)
VALUES (1, 500, to_date('31.07.2015','DD.MM.RRRR'),
'500,00', to_date('14.08.2015','DD.MM.RRRR'));
COMMIT;
```

  Valiku põhjus: Pakub huvi, kuidas mõjutab lisamise kiirust vajadus sünkroonselt hetktõmmist värskendada.

- Ühe tellimuse rea lisamine olemasolevasse tellimusse *Tellimus_rida* tabelis.

```
INSERT INTO Tellimuse_rida (Kaup_ID, Tellimuse_nr,
Myygihind, Kogus) VALUES (500, 50000, '42,15', 1);
COMMIT;
```

  Valiku põhjus: Indeks-organiseeritud tabeli korral on tabeli read sisemiselt võtme järgi sorteeritud. Süsteem peab seda järjekorda säilitama. Seega võiks selle lause täitmine olla optimeeritud disaini korral aeglasem kui tavalise disaini korral.

- Ühe kauba lisamine *Kaup* tabelisse.

```
INSERT INTO Kaup (Kaup_ID, Tootja_ID, Kauba_kood,
Nimetus, Hind, Garantii, Laoseis)
VALUES (1001, 50, 'O74K3194O9', 'Tris Hatdox', '10,00',
'2 aastat', 3);
COMMIT;
```

  Valiku põhjus: Pakub huvi, kuidas mõjutab tabeli sektsioonideks jagamine ja sektsioonideks jagamata indeksid andmete lisamise kiirust.

- Ühe tellimuse seisundi liigi muutmine.

```
UPDATE Tellimus SET Tell_seis_liik_kood=4
WHERE Tellimuse_nr=75000;
COMMIT;
```

  Valiku põhjus: Tellimus oli seisundis „kinnitatud“. Muudatuse tulemusena läheb tellimus seisundisse „tühistatud“. Kinnitatud tellimuste andmed on sektsioonis, kus andmed on pakkimata. Tühistatud tellimuste andmed on sektsioonis, kus andmed on pakitud. Pakub huvi, kuidas mõjutab andmete muutmise kiirust rea liigutamine ühest sektsioonist teise koos pakkimisega.

- Ühe töötaja ametikoha muutmine

```
UPDATE Tootaja SET Ametikoht_kood=2 WHERE Isik_ID=5;
COMMIT;
```

  Valiku põhjus: Pakub huvi, kuidas mõjutab tabelite klasterdamine andmete muutmise kiirust.

Kõik SQL laused on loodud selliselt, et pärast nende käivitamist oleks võimalik võrrelda tavalise andmebaasi tulemusi optimeeritud andmebaasi tulemustega ning leida nende vahelised erinevused.

Kui lausete täitmise kiirustes esineb märkimisväärseid erinevusi, siis kasutatakse AUTOTRACE funktsionaalsust, et leida lausete täitmisplaanid.

Andmemahtude leidmiseks kasutatakse järgmist metoodikat. Lisas 7 kirjeldatud päringu (How do I calculate tables size in Oracle) abil leitakse kõik ühe andmebaasi kasutajaga seotud baastabelid ja hetktõmmised ning nende andmemahtude suurus. Antud päring võtab andmemahtude leidmisel arvesse nii indekseid (*index*) kui ka suuri objekte (*large object*), mistõttu on antud metoodika abil leitud andmemahud võimalikult täpsed.

4. Testandmete genereerimine

Kõik eksperimendi käigus loodud tabelid täidetakse genereeritud testandmetega. Tavalistesse ja optimeeritud tabelitesse sisestatakse ühesugused testandmed, et nende erinevus ei mõjutaks eksperimendi tulemusi.

Genereeritud testandmete hulk on esitatud tabelis 8.

Tabel 8. Genereeritud testandmete hulk ridades

Tabel	Testandmete hulk
Isik	1 010
Töötaja	10
Ametikoht	3
Klient	1 000
Tellimus	100 000
Tellimuse_rida	1 000 000
Tellimuse_seis_liik	5
Kauba_pilt	3 000
Kaup	1 000

Klassifikaatorite tabelite andmed on genereeritud eelnevalt ning sisestatakse tabelitesse käsitsi. Väiksema mahuga tabelite puhul (kuni 1 000 rida) kasutatakse testandmete genereerimiseks veebirakendust *Mockaroo* (http://www.mockaroo.com). Kuna *Mockaroo* tootenimetuste genereerimist ei võimalda, siis kasutatakse kauba nimetuste genereerimiseks veebirakendust *Product Name Generator* (http://online-generator.com/name-generator/product-name-generator.php). Suurema mahuga tabelite puhul (1 000 ja rohkem rida) kasutatakse testandmete genereerimiseks Exceli funktsionaalsusi RAND ja RANDBETWEEN. Kuna *Tellimuse_rida* tabelil on kasutatud kombineeritud primaarvõtit, *PK_Tellimuse_rida*, mis ei võimalda ühe tellimusega siduda kahte samasugust toodet, siis kasutatakse kaupade identifikaatorite ja tellimuse numbrite genereerimiseks Java rakendust *Data generator* (vt lisa 8). Rakendusega genereeritakse vähemalt üks kaup igasse tellimusse ning välditakse olukorda, kus ühes tellimuses oleks rohkem kui üks ühesugune kaup.

Antud eksperimendi raames on vajalik, et tabelisse sisestatud testandmed oleks võimalikud realistlikud. Kuna andmeid on palju, siis on nende genereerimisel tehtud mõningad lihtsustused. Tabelis *Isik* paroole ei soolata, vaid sinna genereeritakse väärtused, mis sarnanevad soolatud paroolile. Tabelis *Kaup* ei kasutata kauba koodi puhul üldlevinud formaati, vaid sinna genereeritakse juhuslikud väärtused. Tabelites *Tellimus* ja *Tellimuse_rida* peaks tellimuses sisalduvate ridade summa vastama tellimuse summale, kuid lihtsuse huvides ei ole andmete genereerimisel seda arvesse võetud.

Kõik genereeritud testandmed on juhusliku pikkuse ja sisuga ning vastavad andmebaasis realiseeritud kitsendustele (nendel testandmetel mille genereerimisel polnud pikkust määratud, on lisatud tabeli veeru pikkus).

- Isik
    - E-meil kuni 50 tähemärki. E-meil võib sisaldada tähti, numbreid, punkte ja kommertsmärki (@).
    - Parool (soolatud) pikkusega 128 tähemärki. Parool võib sisaldada tähti ja numbreid.
    - Eesnimi (inglise keeles) kuni 30 tähemärki. Eesnimi võib sisaldada ainult tähti.
    - Perenimi (inglise keeles) kuni 30 tähemärki. Perenimi võib sisaldada ainult tähti.
    - Aadress (inglise keeles) kuni 100 tähemärki. Aadress võib sisaldada tähti ja numbreid.
    - Telefon kuni 20 tähemärki. Telefoni number võib sisaldada ainult numbreid.
    - Sünniaeg (kuupäev). Sünniaeg ei tohi olla tulevikus.
    - Sool pikkusega 32 tähemärki. Sool võib sisaldada tähti ja numbreid.
- Kauba pilt
    - Faili aadress kuni 50 tähemärki. Faili aadress võib sisaldada tähti, punkti ja kaldkriipsu.

- Kaup
    - Kauba kood kuni 10 tähemärki. Kauba kood võib sisaldada tähti ja numbreid.
    - Nimetus (inglise keeles) kuni 100 tähemärki. Nimetus võib sisaldada ainult tähti.
    - Hind kuni 5 numbrimärki 2 kohaga peale koma.
    - Garantii pikkusega 8 tähemärki. Garantii võib sisaldada tähti ja numbreid.
    - Laoseis täisarv vahemikus 0-10 (otspunktid kaasa arvatud).
- Tellimus
    - Tellimuse seisundi liigi kood täisarv vahemikus 1 kuni 5 (otspunktid kaasa arvatud).
    - Esitamise kuupäev (kuupäev). Esitamise kuupäev ei tohi olla peale maksetähtaega.
    - Summa kuni 6 numbrimärki 2 kohaga peale koma.
    - Maksetähtaeg (kuupäev). Maksetähtaeg ei tohi olla enne esitamise kuupäeva.
- Tellimuse rida
    - Müügihind kuni 5 numbrimärki 2 kohaga peale koma.
    - Kogus täisarv vahemikus 0-10 (otspunktid kaasa arvatud).
- Tootja
    - Registri number kuni 8 tähemärki. Registri number võib sisaldada tähti ja numbreid.
    - Nimi (inglise keeles) kuni 50 tähemärki. Nimi võib sisaldada ainult tähti.

Andmete lisamiseks andmebaasi kasutatakse Oracle SQL Developeri sisseehitatud funktsionaalsust *Data Import Wizard*. Selle käivitamisel viidatakse andmeid sisaldava faili asukohale, valitakse välja sealt sobivad veerud ning sisestatakse need andmebaasi. Kuigi enamik andmeid on salvestatud Exceli formaati (*.XLS*), siis suurema mahuga tabelite andmed on salvestatud komaeraldusega väärtustena (*.CSV*). Nii on võimalik vältida piirangut Oracle SQL Developeri poolt kasutatavale mälule, kus Java virtuaalmasinale eraldatava mälu suurus jääb üldjuhul alla 1024MB (Configuring memory usage in Oracle SQL Developer), ning sisestada andmebaasi ka need andmed.

Andmete lisamise kiirendamiseks loodi kõigepealt ainult baastabelid. Seejärel lisati baastabelitesse testandmed. Seejärel loodi täiendavad indeksid ja hetktõmmised. Kõige lõpuks värskendati statistikat.

5. Eksperimendi tulemused

Eksperimendi tulemused on jaotatud kolme tabelisse. Esimeses ja teises tabelis on esitatud päringud ja andmemuudatused koos neile vastavate täitmiskiirustega. Kolmandas tabelis on välja toodud baastabelite ja hetktõmmiste andmemahud.

Iga päringu ja andmemuudatuse kohta on esitatud nii tavalise kui ka optimeeritud disainiga tabeli tulemused. Selleks, et edaspidi eksperimendi tulemuste analüüsis päringutele ja andmemuudatustele viidata, on need vastavalt nummerdatud.

Tabelis 9 on esitatud päringute keskmised täitmiskiirused sekundites. Väärtuse leidmiseks on välja arvutatud kolme mõõtmise aritmeetiline keskmine. Disain, kus täitmiskiirus oli parem, on märgitud ära rasvases kirjas.

Tabel 9. Katses kasutatud päringute keskmised täitmiskiirused

Päring	Disaini tüüp	Keskmine täitmiskiirus
1	`SELECT Faili_aadress FROM Kauba_pilt WHERE Kaup_ID = 500;`	
	TAV	0.942
	OPT	**0.915**
2	`SELECT K.kaup_id, Count(*) cnt FROM Kaup K INNER JOIN Kauba_pilt KP ON K.kaup_id=KP.kaup_id WHERE K.hind<25 GROUP BY K.kaup_id;`	
	TAV	1.184
	OPT	**1.054**
3	`SELECT * FROM Tellimus WHERE Tell_seis_liik_kood = 4;`	
	TAV	4.482
	OPT	**3.066**
4	`SELECT SUM(Myygihind) FROM Tellimuse_rida WHERE Tellimuse_nr BETWEEN 24999 AND 25001;`	
	TAV	1.020
	OPT	**0.882**
5	`SELECT T.Isik_ID, I.Eesnimi, I.Perenimi, A.Nimetus Amet FROM (Tootaja T INNER JOIN Isik I ON T.Isik_ID=I.Isik_ID) INNER JOIN Ametikoht A ON T.Ametikoht_kood=A.Ametikoht_kood;`	

Päring	Disaini tüüp	Keskmine täitmiskiirus
	TAV	**0.906**
	OPT	0.932
6	`SELECT COUNT(*) Arv FROM Kaup WHERE Hind BETWEEN 1 AND 35;`	
	TAV	**0.897**
	OPT	0.912
7	`SELECT * FROM Tell_detail WHERE Tellimuse_nr=50000;`	
	TAV	**0.929**
	OPT	1.020
8	`SELECT Tellimuse_nr, Nimetus AS Seisund FROM Tellimus T INNER JOIN Tellimuse_seis_liik TSL ON TSL.Tell_seis_liik_kood=T.Tell_seis_liik_kood;`	
	TAV	1.974
	OPT	**1.715**

Tabelis 10 on esitatud andmemuudatuste täitmiskiirused sekundites. Väärtuse leidmiseks on kokku liidetud lause täitmiseks kui tulemuse kinnitamiseks kulunud aeg. Disain, kus täitmiskiirus oli parem, on märgitud ära rasvases kirjas.

Tabel 10. Katses kasutatud andmemuudatuste täitmiskiirused

Andmemuudatus	Disaini tüüp	Täitmiskiirus
1	`INSERT INTO Tellimus (Tell_seis_liik_kood, Isik_ID, Esitamise_kuupaev, Summa, Maksetahtaeg) VALUES (1, 500, to_date('31.07.2015','DD.MM.RRRR'), '500,00', to_date('14.08.2015','DD.MM.RRRR')); COMMIT;`	
	TAV	**1.853**
	OPT	2.425
2	`INSERT INTO Tellimuse_rida (Kaup_ID, Tellimuse_nr, Myygihind, Kogus) VALUES (500, 50000, '42,15', 1); COMMIT;`	
	TAV	2.023
	OPT	**1.891**

Andmemuudatus	Disaini tüüp	Täitmiskiirus
3	`INSERT INTO Kaup (Kaup_ID, Tootja_ID, Kauba_kood, Nimetus, Hind, Garantii, Laoseis) VALUES (1001, 50, 'O74K3194O9', 'Tris Hatdox', '10,00', '2 aastat', 3); COMMIT;`	
	TAV	**1.782**
	OPT	1.904
4	`UPDATE Tellimus SET Tell_seis_liik_kood=4 WHERE Tellimuse_nr=75000; COMMIT;`	
	TAV	**1.835**
	OPT	2.066
5	`UPDATE Tootaja SET Ametikoht_kood=2 WHERE Isik_ID=5; COMMIT;`	
	TAV	1.914
	OPT	**1.849**

Tabelis 11 on esitatud baastabelite ja hetktõmmiste andmemahtude suurused megabaitides. Kuna vaadete puhul arvutatakse väärtused jooksult vaate aluseks olevate tabelite põhjal kui kasutaja vaate poole pöördub, siis ei ole neid antud tabelis esitatud. Iga disaini puhul on vastavate tabelite paaris märgitud rasvase kirjaga tabel, kus andmemahtude suurus on väiksem. Kui andmemahud on ühesugused, siis rasvast kirja ei kasutata.

Tabel 11. Baastabelite ja hetktõmmiste andmemahtude suurused

Tabel	Andmemahu suurus (MB)
TAV_TELLIMUSE_RIDA	77
OPT_TELLIMUSE_RIDA	**70**
TAV_TELLIMUS	**13**
OPT_TELLIMUS	64
TAV_KAUBA_PILT	**0.69**
OPT_KAUBA_PILT	32.63
TAV_KAUP	**0.38**
OPT_KAUP	32.25

Tabel	Andmemahu suurus (MB)
OPT_TELL_DETAIL	32
OPT_TELL_NIMEK	13
TAV_ISIK	0.5
OPT_ISIK	0.5
TAV_TOOTJA	0.19
OPT_TOOTJA	0.19
TAV_AMETIKOHT	0.19
OPT_AMETIKOHT	**0.13**
TAV_TELLIMUSE_SEIS_LIIK	0.19
OPT_TELLIMUSE_SEIS_LIIK	**0.13**
TAV_KLIENT	0.13
OPT_KLIENT	0.13
TAV_TOOTAJA	0.19
OPT_TOOTAJA	**0.13**

6. Eksperimendi tulemuste analüüs ja järeldused

Järgnevalt on esitatud eksperimendis käivitatud päringute ja andmemuudatuste iga mõõtmise täitmiskiirused. Päringute puhul mõõdetakse täitmiskiirust kolm korda. Kuna andmemuudatuste puhul mõõdetakse täitmiskiirusi ainult ühe korra, siis nende puhul on eraldi välja toodud lause täitmiseks ja tulemuse kinnitamiseks kulunud aeg. Nende päringute või andmemuudatuste puhul, kus täitmiskiirused erinesid oluliselt või kus tulemused ei vastanud ootustele, on esitatud ka lausete täitmisplaanid.

6.1. Päringute täitmiskiirused

Tabelis 12 on esitatud 1. päringu täitmiskiirused. Päringus leiti ühe kaubaga seotud piltide aadressid *Kauba_pilt* tabelist.

Tabel 12. 1. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	0.968	0.93	0.928	0.942
OPT	0.961	0.917	0.867	0.915

Tavalise disainiga andmebaasis pidi süsteem kasutama kogu tabelile loodud indeksit veerule *kaup_id*. Oracle ei indekseeri vaikimisi välisvõtme veerge, kuid antud töös on tavalise disaini korral kõigile välisvõtmetele loodud indeksid, eeldusel, et need ei dubleeri tabelile automaatselt loodud indekseid. Optimaalse disaini korral oli süsteemil võimalik lugeda lokaalset sektsioonideks jagatud indeksit veerule *kaup_id*, mis on mahult väiksem ning seega leiti kiiremini viide plokile, milles asuvad huvipakkuva kauba andmed.

Tabelis 13 on esitatud 2. päringu täitmiskiirused. Päringus leiti odavate kaupadega seotud piltide arv.

Tabel 13. 2. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	1.419	1.08	1.052	1.184
OPT	1.064	1.044	1.054	1.054

Kuna vaja oli kokku ühendada kaks tabelit ning mõlemast tabelist lugeda andmeid vaid ühest sektsioonist, siis oli optimeeritud disaini korral päringu täitmine kiirem kui tavalise disaini korral. Tabeli *Kauba_pilt* puhul piisas süsteemil vaid veerule *kaup_id* loodud indeksi lugemisest ning tabelit ennast ei olnud vaja puutuda. Seega kiirendas optimeeritud disaini korral päringut tabelis *Kauba_pilt* välisvõtmele loodud lokaalne sektsioonideks jaotatud indeks, sest see on mahult väiksem kui tavalise disaini puhul välisvõtme veerule loodud indeks, mis hõlmab tervet tabelit.

Tabelis 14 on esitatud 3. päringu täitmiskiirused. Päringus leiti kõik „esitatud“ seisundis tellimused *Tellimus* tabelist.

Tabel 14. 3. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	7.401	3.204	2.841	4.482
OPT	3.278	2.816	3.104	3.066

Mõlema disaini korral kasutas süsteem lause täitmiseks tabeli täielikku läbiskanneerimist (*full table scan*). Kuna optimeeritud disaini korral tuli andmeid lugeda ainult ühest sektsioonist, siis oli päringu täitmine kiirem kui tavalise disaini korral. Lisaks sellele olid optimeeritud disainiga andmebaasis andmed pakitud ning loetavate plokkide hulk väiksem.

Tabelis 15 on esitatud 4. päringu täitmiskiirused. Päringuga leiti mitme tellimusega seotud kaupade hind *Tellimuse_rida* tabelist.

Tabel 15. 4. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	1.237	0.931	0.892	1.020
OPT	0.882	0.903	0.862	0.882

Kuna optimeeritud disaini korral olid tänu indeks-orienteeritud tabelile lähestikku paiknevate numbritega tellimuste read salvestatud kettale füüsiliselt lähestikku ning müügihinnad olid selles samas indeksi struktuuris, mitte eraldi tabeliplokkides, siis oli päringu täitmine kiirem kui tavalise disaini korral. Tavalise disaini puhul pidi süsteem

kõigepealt lugema indeksiplokke ja siis tabeliplokke, mistõttu kulus andmete lugemiseks rohkem aega.

Tabelis 16 on esitatud 5. päringu täitmiskiirused. Päringuga leiti kõik töötajad ja nende ametikohad.

Tabel 16. 5. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	0.913	0.925	0.881	0.906
OPT	0.995	0.922	0.878	0.932

Kuigi optimeeritud disaini korral olid töötajate ja ametite andmed samas klastris ning päringu täitmine oleks võinud olla kiirem kui tavalise disaini korral, siis antud eksperimendis olid optimeeritud disaini korral täitmiskiirused suuremad.

Selleks, et jõuda põhjuste jälile, on Tabelites 17 ja 18 esitatud nii tavalise kui ka optimeeritud disainiga andmebaasis läbi viidud päringute täitmisplaanid.

Tabel 17. 5. päringu täitmisplaan tavalise disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	370	14 (0)	00:00:01
1	NESTED LOOPS					
2	NESTED LOOPS		10	370	14 (0)	00:00:01
3	HASH JOIN		10	190	4 (0)	00:00:01
4	VIEW	index$_join$_004	3	39	2 (0)	00:00:01
5	HASH JOIN					
6	INDEX FAST FULL SCAN	TAV_PK_AMETIK_KOOD	3	39	1 (0)	00:00:01
7	INDEX FAST FULL SCAN	TAV_TC_AMETIK_NIM	3	39	1 (0)	00:00:01
8	VIEW	index$_join$_001	10	60	2 (0)	00:00:01
9	HASH JOIN					
10	INDEX FAST FULL SCAN	TAV_PK_TOOT_ISIK	10	60	1 (0)	00:00:01
11	INDEX FAST FULL	TAV_TC_AMETIK_	10	60	1 (0)	00:00:01

	SCAN	KOOD				
12	INDEX UNIQUE SCAN	TAV_PK_ISIK	1		0 (0)	00:00:01
13	TABLE ACCESS BY INDEX ROWID	TAV_ISIK	1	18	1 (0)	00:00:01

Tabel 18. 5. päringu täitmisplaan optimeeritud disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	370	14 (0)	00:00:01
1	NESTED LOOPS					
2	NESTED LOOPS		10	370	14 (0)	00:00:01
3	HASH JOIN		10	190	4 (0)	00:00:01
4	VIEW	index$_join$_004	3	39	2 (0)	00:00:01
5	HASH JOIN					
6	INDEX FAST FULL SCAN	OPT_PK_AMETIK_KOOD	3	39	1 (0)	00:00:01
7	INDEX FAST FULL SCAN	OPT_TC_AMETIK_NIM	3	39	1 (0)	00:00:01
8	VIEW	index$_join$_001	10	60	2 (0)	00:00:01
9	HASH JOIN					
10	INDEX FAST FULL SCAN	OPT_PK_TOOT_ISIK	10	60	1 (0)	00:00:01
11	INDEX FAST FULL SCAN	OPT_TC_AMETIK_KOOD	10	60	1 (0)	00:00:01
12	INDEX UNIQUE SCAN	OPT_PK_ISIK	1		0 (0)	00:00:01
13	TABLE ACCESS BY INDEX ROWID	OPT_ISIK	1	18	1 (0)	00:00:01

Nagu tabelitest selgub, siis on mõlema disaini korral päringute täitmisplaanid ühesugused. Põhjus võib peituda selles, et andmete väikese hulga tõttu ei näinud andmebaasisüsteem erinevate tabelite põhjal erinevat lähenemist proovida.

Tabelis 19 on esitatud 6. päringu täitmiskiirused. Päringuga leiti ühes hinnavahemikus olevate kaupade arv.

Tabel 19. 6. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	0.912	0.895	0.883	0.897
OPT	0.927	0.922	0.886	0.912

Kuigi andmeid tuli lugeda ainult kahest sektsioonist ning päringu täitmine oleks võinud optimeeritud disaini korral olla kiirem kui tavalise disaini korral, siis antud eksperimendis olid optimeeritud disaini korral täitmiskiirused suuremad.

Selleks, et jõuda põhjuste jälile, on Tabelites 20 ja 21 esitatud nii tavalise kui ka optimeeritud disainiga andmebaasis läbi viidud päringute täitmisplaanid.

Tabel 20. 6. päringu täitmisplaan tavalise disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	4	5 (0)	00:00:01
1	SORT AGGREGATE		1	4		
2	TABLE ACCESS FULL	TAV_KAUP	344	1376	5 (0)	00:00:01

Tabel 21. 6. päringu täitmisplaan optimeeritud disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		1	4	27 (0)	00:00:01		
1	SORT AGGREGATE		1	4				
2	PARTITION RANGE ITERATOR		344	1376	27 (0)	00:00:01	1	2
3	TABLE ACCESS FULL	OPT_KAUP	344	1376	27 (0)	00:00:01	1	2

PARTITION RANGE ITERATOR operatsioon optimeeritud disainiga andmebaasi päringu korral näitab, et andmebaasisüsteem vaatab lause täitmiseks sektsioonid, mis vastavad päringu tingimuses esitatud hinnavahemikule, kuid mitte kõik tabeli

sektsioonid (seda näitaks *PARTITION RANGE ALL*). Antud juhul on andmebaasisüsteemil vaja läbi vaadata ainult kaks sektsiooni. Seega on kõik nii nagu soovitud, kuid optimeeritud disaini puhul ei tinginud see paremat täitmise kiirust. Ilmselgelt oli ridade arv tabelis *Kaup* (1000) liiga väike, et selle sektsioonideks jagamise eelised välja tuleksid. Kindlasti kulub süsteemil ka mingi aeg ühe sektsiooni juurest teise juurde liikumiseks ning kui andmeid on vähe, siis see „sööb“ ära üksikute sektsioonide lugemisest tuleneva ajalise võidu.

Tabelis 22 on esitatud 7. päringu täitmiskiirused. Päringuga kuvati ühes tellimuses sisalduvate toodete nimekiri vaate või hetktõmmise *Tell_detail* põhjal.

Tabel 22. 7. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	0.998	0.903	0.887	0.929
OPT	**1.08**	**0.98**	**1.001**	**1.020**

Kuigi hetktõmmise korral on väärtused eelnevalt välja arvutatud ja salvestatud ning päringu täitmine oleks võinud olla kiirem hetktõmmise kui vaate korral, siis olid optimeeritud disaini korral täitmiskiirused suuremad ning hetktõmmis ei andnud suurt ajalist võitu.

Selleks, et jõuda põhjuste jälile, on Tabelites 23 ja 24 esitatud nii tavalise kui ka optimeeritud disainiga andmebaasis läbi viidud päringute täitmisplaanid.

Tabel 23. 7. päringu täitmisplaan tavalise disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	320	18 (0)	00:00:01
1	HASH JOIN		10	320	18 (0)	00:00:01
2	TABLE ACCESS BY INDEX ROWID BATCHED	TAV_TELLIMUSE_RIDA	10	160	13 (0)	00:00:01
3	INDEX RANGE SCAN	TAV_PK_TELLIMUSE_RIDA	10		3 (0)	00:00:01
4	TABLE ACCESS FULL	TAV_KAUP	1000	16000	5 (0)	00:00:01

Tabel 24. 7. päringu täitmisplaan optimeeritud disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	240	1102 (1)	00:00:01
1	MAT_VIEW ACCESS FULL	OPT_TELL_DETAIL	10	240	1102 (1)	00:00:01

Nagu tabelitest selgub, siis optimeeritud disaini korral hetktõmmisest andmete pärimisel täiendavaid operatsioone teha vaja ei olnud. Kuna tavalise disainiga andmebaasis on aga täitmiskiirused veidi suuremad, siis tuleb jällegi tõdeda, et sellise vaate alampäringu ja selliste andmemahtude puhul nagu töös kasutati ei olnud hetktõmmise järgi vajadust.

Tabelis 25 on esitatud 8. päringu täitmiskiirused. Päringuga leiti kõigi tellimuste numbrite ja seisundite nimed.

Tabel 25. 8. päringu täitmiskiirused

Disaini tüüp	1. täitmiskiirus	2. täitmiskiirus	3. täitmiskiirus	Keskmine täitmiskiirus
TAV	2.123	2.13	1.669	1.974
OPT	1.785	1.71	1.651	1.715

Kuna optimeeritud disaini korral oli täitmiskiirus parem kui tavalise disaini korral, siis üks selgitus on, et ühe tabeli räsiklaster võimaldas andmeid tabelist kiiremini lugeda. See oli tingitud sellest, et optimeeritud disaini puhul piisas ühe tabeli räsiklastril vaid primaarvõtme põhjal räsiväärtuse leidmisest, mis viitas tabeli rea füüsilisele asukohale. Tavalise disaini korral kulus aga indeksi põhjal rea leidmiseks rohkem lugemisi (Single Table Hash Clusters Demystified). Samas on optimeeritud disaini korral tabel *Tellimus* sektsioonideks jagatud ja osa sektsioone pakitud ja ka see võiks mõju avaldada. Pakitud sektsioonide korral tuleb süsteemil teoorias lugeda vähem plokke ja see võiks töökiirust parandada. Samas kui vaadata loogiliste (mälust lugemiste) arvu (*consistent gets*), siis on see optimaalse disaini korral suurem (8341) kui tavalise disaini korral (7228), mis on eelmise mõttega vastuolus ning ühtlasi eeldaks, et tavalise disaini puhul oleks täitmiskiirus parem kui optimaalse disaini puhul.

6.2. Andmemuudatuste täitmiskiirused

Tabelis 26 on esitatud 1. andmemuudatuse täitmiskiirused. Andmemuudatusega lisati üks tellimus.

Tabel 26. 1. andmemuudatuse täitmiskiirused

Disaini tüüp	Andmemuudatuse täitmiskiirus	Kinnitamise täitmiskiirus	Täitmiskiiruste summa
TAV	0.927	0.926	1.853
OPT	0.993	1.432	2.425

Kuna optimeeritud disaini korral oli täitmiskiirus aeglasem kui tavalise disaini korral, siis võib öelda, et vajadus sünkroonselt hetktõmmist värskendada aeglustab andmete lisamise kiirust.

Tabelis 27 on esitatud 2. andmemuudatuse täitmiskiirused. Andmemuudatusega lisati üks tellimuse rida olemasolevasse tellimusse *Tellimus_rida* tabelis.

Tabel 27. 2. andmemuudatuse täitmiskiirused

Disaini tüüp	Andmemuudatuse täitmiskiirus	Kinnitamise täitmiskiirus	Täitmiskiiruste summa
TAV	1.049	0.974	2.023
OPT	**0.963**	**0.928**	**1.891**

Kuigi optimeeritud disaini korral on indeks-orienteeritud tabelis read sisemiselt võtme järgi sorteeritud ning andmemuudatuse täitmine oleks võinud optimeeritud disaini korral olla aeglasem kui tavalise disaini korral, siis antud eksperimendis olid optimeeritud disaini korral täitmiskiirused väiksemad.

Selleks, et jõuda põhjuste jälile, on Tabelites 28 ja 29 esitatud nii tavalise kui ka optimeeritud disainiga andmebaasis läbi viidud andmemuudatuste täitmisplaanid.

Tabel 28. 2. andmemuudatuse täitmisplaan tavalise disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	INSERT STATEMENT		1	16	1 (0)	00:00:01
1	LOAD TABLE CONVENTIONAL	TAV_TELLIMUSE_RIDA				

Tabel 29. 2. andmemuudatuse täitmisplaan optimeeritud disainiga andmebaasis

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	INSERT STATEMENT		1	16	1 (0)	00:00:01
1	LOAD TABLE CONVENTIONAL	OPT_TELLIMUSE_RIDA				

Nagu tabelitest selgub, siis on mõlema disaini korral andmemuudatuste täitmisplaanid ühesugused. Tavalise disaini korral tuli lisada rida piisava vaba ruumiga tabeliplokki ning uuendada tabelile loodud kahte indeksit. Indeks-organiseeritud tabeli korral on tabeli andmed primaarvõtme alusel loodud indeksi struktuuris – seega polnud vaja lisada andmeid eraldi tabeli plokkidesse ja primaarvõtme indeksi plokkidesse ning kokkuvõttes oli töökiirus parem.

Tabelis 30 on esitatud 3. andmemuudatuse täitmiskiirused. Andmemuudatusega lisati üks kaup *Kaup* tabelisse.

Tabel 30. 3. andmemuudatuse täitmiskiirused

Disaini tüüp	Andmemuudatuse täitmiskiirus	Kinnitamise täitmiskiirus	Täitmiskiiruste summa
TAV	0.865	0.917	1.782
OPT	0.996	0.908	1.904

Kuna optimeeritud disaini korral oli täitmiskiirus halvem kui tavalise disaini korral, siis võib öelda, et sektsioonideks jagamine ja sektsioonideks jagamata indeks vähendavad andmete lisamise kiirust.

Tabelis 31 on esitatud 4. andmemuudatuse täitmiskiirused. Andmemuudatusega muudeti ühe tellimuse seisundi liiki.

Tabel 31. 4. andmemuudatuse täitmiskiirused

Disaini tüüp	Andmemuudatuse täitmiskiirus	Kinnitamise täitmiskiirus	Täitmiskiiruste summa
TAV	0.919	0.916	1.835
OPT	1.084	0.982	2.066

Kuna optimeeritud disaini korral oli täitmiskiirus halvem kui tavalise disaini korral, siis võib öelda, et rea liigutamine ühest sektsioonist teise koos pakkimisega vähendab andmete muutmise kiirust. Kuigi veerule oli lisatud globaalne sektsioonideks jaotatud indeks, siis see andmemuudatust ei kiirendanud, kuna selle põhjal pakkimata sektsioonist andmete leidmine nõudis täiendavat operatsiooni.

Tabelis 32 on esitatud 5. andmemuudatuse täitmiskiirused. Andmemuudatusega muudeti ühe töötaja ametikohta.

Tabel 32. 5. andmemuudatuse täitmiskiirused

Disaini tüüp	Andmemuudatuse täitmiskiirus	Kinnitamise täitmiskiirus	Täitmiskiiruste summa
TAV	0.976	0.938	1.914
OPT	0.928	0.921	1.849

Kuna optimeeritud disaini korral oli täitmiskiirus parem kui tavalise disaini korral, siis võib oletada, et tabelite klasterdamine parandab andmete muutmise kiirust (Tables and Table Clusters). Antud tabelites kasutatav räsiklaster oli loodud veeru *Ametikoht_kood* põhjal, mistõttu töötajale vastava ametikoha muutmisel leiti ametikoha koodile vastav räsiväärtus. Leitud räsiväärtus viitas aga tabeli rea füüsilisele asukohale, mistõttu oli andmete leidmine muudatuse läbiviimiseks kiirem.

6.3. Andmemahud

Tabelis 11 esitatud baastabelite ja hetktõmmiste andmemahtude põhjal on näha, et üldjuhul on tavalise disaini korral andmemahud väiksemad kui optimeeritud disaini korral. Kuna tavalise disainiga andmebaasis kasutati hetktõmmiste asemel vaateid, kus väärtused arvutatakse jooksult kui kasutaja vaate poole pöördub, siis ei olnud vaja nendes sisalduvaid andmeid eraldi salvestada. Seetõttu pole neid ka tabelis ning andmemahud on selle võrra väiksemad.

Tabelites *Tellimus, Kauba_pilt* ja *Kaup* on optimeeritud disaini korral andmemahud suuremad kui tavalise disaini korral. Kuna optimeeritud disainiga tabelites on kasutatud sektsioonideks jagamist, siis on sellega seotud informatsiooni salvestamiseks vaja täiendavat ruumi, mistõttu on ka andmemahud suuremad.

Tabelites *Tellimuse_rida*, *Ametikoht*, *Tellimuse_seis_liik* ja *Töötaja* on optimeeritud disaini korral andmemahud väiksemad kui tavalise disaini korral. Tabel *Tellimuse_rida* on loodud indeks-orienteeritud tabelina. Kuna indeks-orienteeritud tabelis ei ole võtmeveergusid indeksisse kopeeritud ning rea identifikaatorite salvestamiseks ei ole eraldi ruumi tarvis, siis on ka andmemahud selle võrra väiksemad (Speeding Up Index Access with Index-Organized Tables). Tabelid *Ametikoht*, *Tellimuse_seis_liik* ja *Töötaja* on aga koondatud kas sisemisel tasemel või omavahel räsiklastrisse. Kuna klastri puhul on ühesugused andmed paigutatud ühte plokki ning klastri võtmeväärtus on salvestatud ühekordselt, siis on ka andmemahud väiksemad (Advantages of Clustered Tables in Oracle).

Tabelites *Isik, Tootja ja Klient* on mõlema disaini korral andmemahud võrdsed, kuna nendes tabelites optimeerimisvõimalusi ei kasutatud.

6.4. Järeldused

Vastavalt ülaltoodud analüüsile võib järeldada, et alati ei pruugi andmebaasis kasutatavad optimeerimisvõimalused anda oodatud tulemust. See kas optimeerimisvõimalused päringute ja andmemuudatuste täitmiskiiruseid parandavad ning andmemahtusid vähendavad sõltub sellest, milline on andmebaasi struktuur ning kas optimeerimisvõimalusi on tabelites õigesti kasutatud.

Nagu päringute ja andmemuudatuste puhul oli näha, siis üldjuhul parandasid indeks-orienteeritud, sektsioonideks jagatud ja klastrisse koondatud tabelid päringute täitmiskiiruseid. Erandiks oli andmete pärimine mitme sektsiooni põhjal, kus ridade lugemisel oli sektsioonis olevate andmete töötlemiseks vaja täiendavat operatsiooni ning kuna loetavaid andmeid oli vähe, siis see nullis väiksema hulga andmete lugemisest saadud töökiiruse võidu. Samuti olid sektsioonideks jagatud tabelite puhul aeglasemad andmemuudatused, kus sektsioonidesse andmete lisamine võtab rohkem aega.

Andmemahtude põhjal oli aga näha, et tavalise disainiga tabelid võtavad enda alla vähem salvestusruumi kui optimeeritud disainiga tabelid. Kuigi klastrisse jaotatud tabelite korral olid andmemahud väiksemad, siis sektsioonideks jagatud tabelite korral olid need oluliselt suuremad. Seega võib öelda, et optimeeritud disainiga

tabelites läbi viidud päringute ja andmemuudatuste paremad täitmiskiirused on saavutatud täiendava andmemahu arvelt.

Töö piiranguks võib lugeda asjaolu, et mitmete optimeerimismeetodite eeliste ja probleemide väljatoomiseks oli testandmeid liiga vähe. Samuti oli optimeeritud disaini korral mitmetel tabelitel kasutusel korraga mitu optimeerimismeetodit, mis ei võimaldanud selgelt aru saada, milles täpselt peitub mõne töökiiruse muudatuse põhjus. Töö edasiseks edasiarenduseks võiks olla veel mahukam eksperiment, mis nende piirangutega arvestaks.

Kuna eksperimendis kasutatav andmebaas oli loodud vastavalt konkreetse infosüsteemi nõuetele, siis ei pruugi antud optimeerimisvõimalused anda teistes infosüsteemides sarnaseid tulemusi. Samuti ei olnud kasutatud kõiki optimeerimisvõimalusi, mistõttu võivad teistsugused meetodid anda hoopis paremaid tulemusi.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli Oracle tabelite optimeerimise meetodite uurimine ning nende mõju hindamine päringute ja andmemuudatuste kiirusele. Selle saavutamiseks võrreldi kahte erinevat SQL-andmebaasi disaini, milles oli erinev tabelite sisemine struktuur. Ühes disainis kasutati tavalisi baastabeleid ja nende peale loodud vaateid, teises aga erinevaid tabelite loomiseks pakutavaid optimeerimisvõimalusi ning hetktõmmiseid. Erinevalt tavalisest disainist, kasutati optimeeritud disaini korral indeks-orienteeritud ja klastrisse koondatud tabeleid, tabelite sektsioonideks jagamist, lokaalset indeksit ning hetktõmmist. Andmebaasid täideti ühesuguste testandmetega ning mõlemas andmebaasis viidi läbi ühesugused päringuid ja andmemuudatused. Lõpuks võrreldi mõõdetud täitmiskiiruseid ning jõuti järeldusele optimeerimisvõimaluste mõju kohta andmebaasi jõudlusele.

Eksperimendi tulemusena selgus, et optimeeritud disainiga tabelites olid päringute täitmiskiirused üldiselt paremad kui tavalistes tabelites, kuid mitmel juhul olid andmemuudatused aeglasemad. Andmemahud olid aga väiksemad tavaliste disainiga andmebaasis, mistõttu võib väita, et optimeeritud disainiga andmebaasis on päringute paremad täitmiskiirused saavutatud täiendava andmemahu arvelt. Seega, nagu alati, nõuab eelise saamine ühes valdkonnas ohverdusi kuskil mujal.

Lõpetuseks sooviksin tänada oma juhendajat Erki Eessaart.

Summary

The goal of this thesis was to examine different optimization methods of Oracle database tables and their impact on the performance of queries and data modifications. To achieve this, two SQL databases with different designs, which had varying internal table structures, were compared. One design used heap-organized tables and ordinary views on top of them, while the other used different optimization methods of tables and snapshots. Apart from the conventional design, optimized design used index-organized and clustered tables, partitioning, local index, and materialized views. Both databases were filled with identical test data. In both databases the same queries and data modifications were carried out. Finally, the results were compared and the impact of optimized tables on the performance was concluded.

The results of the experiment showed that the queries executed based on the optimized tables had generally better performance than queries based on ordinary, heap-organized tables. However, in multiple cases data modifications were slower based on the optimized tables. Data volumes were lower in heap-organized tables. Thus, it can be concluded that database with the optimized design has better query performance at the expense of additional data volumes. Thus, as always, gaining an advantage in some aspect requires sacrifices somewhere else.

Finally, I would like to thank my supervisor Erki Eessaar.

Kasutatud kirjandus

1. Advantages of Clustered Tables in Oracle. [WWW] http://www.relationaldbdesign.com/extended-database-features/module3/clustered-tables-advantages.php (8.05.2015)
2. Bad Hash Clusters. [WWW] http://www.orafaq.com/tuningguide/bad%20hash%20cluster.html (11.03.2015)
3. Burelson, D. (2010) Oracle Global Index vs. Local Index. May 14, 2010. [WWW] http://www.dba-oracle.com/t_global_local_partitioned_index.htm (23.05.2015)
4. Configuring memory usage in Oracle SQL Developer. [WWW] http://www.thatjeffsmith.com/archive/2014/06/configuring-memory-usage-in-oracle-sql-developer/ (7.04.2015)
5. DB-Engines Ranking. May 2015. [WWW] http://db-engines.com/en/ranking (23.05.2015)
6. How do I calculate tables size in Oracle. [WWW] http://stackoverflow.com/questions/264914/how-do-i-calculate-tables-size-in-oracle (18.03.2015)
7. Kyte, T. (2010). Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions, Second Edition. Ameerika Ühendriigid: Springer Science+Business Media.
8. Materialized Views in Oracle. [WWW] http://oracle-base.com/articles/misc/materialized-views.php (17.05.2015)
9. Materialized View Concepts and Architecture. [WWW] http://docs.oracle.com/cd/B10501_01/server.920/a96567/repmview.htm (17.05.2015)
10. Sams, R., Rodionov, D., Sapožnikov A. (2014) Arvutitoodete internetipood, Andmebaasid I IDU0220. Tallinna Tehnikaülikool, Tallinn.
11. Sams, R., Rodionov, D. (2014) Arvutitoodete internetipood, Andmebaasid II IDU0230. Tallinna Tehnikaülikool, Tallinn.
12. Single Table Hash Clusters Demystified. [WWW] http://oracleinaction.com/single-tab-hash-cluster/ (5.05.2015)
13. Speeding Up Index Access with Index-Organized Tables. [WWW] http://docs.oracle.com/cd/B10501_01/appdev.920/a96590/adg07iot.htm (8.05.2015)

14. Tables and Table Clusters. [WWW]
https://docs.oracle.com/cd/E11882_01/server.112/e40540/tablecls.htm#CNCPT609 (6.05.2015)

Lisad

Lisa 1 - tavaliste tabelitega andmebaasi loomise laused

```
CREATE TABLE TAV_Tootaja (
     Isik_ID NUMBER ( 10 ) NOT NULL,
     Ametikoht_kood NUMBER ( 2 ) NOT NULL,
     CONSTRAINT TAV_PK_Toot_Isik PRIMARY KEY (Isik_ID)
     );

CREATE TABLE TAV_Klient (
     Isik_ID NUMBER ( 10 ) NOT NULL,
     CONSTRAINT TAV_PK_Klient_Isik PRIMARY KEY (Isik_ID)
     );

CREATE TABLE TAV_Tellimuse_rida (
     Kaup_ID NUMBER ( 10 ) NOT NULL,
     Tellimuse_nr NUMBER ( 10 ) NOT NULL,
     Myygihind NUMBER ( 10, 2 ) NOT NULL,
     Kogus NUMBER ( 10 ) NOT NULL,
     CONSTRAINT TAV_PK_Tellimuse_rida PRIMARY KEY
(Tellimuse_nr,
          Kaup_ID),
     CONSTRAINT TAV_TC_Tell_r_mh CHECK (Myygihind > 0),
     CONSTRAINT TAV_TC_Tell_r_kog CHECK (Kogus > 0)
     );

CREATE TABLE TAV_Kauba_pilt (
     Kauba_pilt_ID NUMBER ( 10 ) NOT NULL,
     Kaup_ID NUMBER ( 10 ) NOT NULL,
     Faili_aadress VARCHAR2 ( 50 ) NOT NULL,
     CONSTRAINT TAV_TC_Kauba_pilt UNIQUE (Faili_aadress),
     CONSTRAINT TAV_PK_Kauba_pilt PRIMARY KEY (Kauba_pilt_ID),
     CONSTRAINT TAV_TC_Kauba_pilt_Fn CHECK (NOT
          REGEXP_LIKE(Faili_aadress, '^[[:space:]]*$'))
     );

CREATE TABLE TAV_Tellimuse_seis_liik (
     Tell_seis_liik_kood NUMBER ( 2 ) NOT NULL,
     Nimetus VARCHAR2 ( 50 ) NOT NULL,
     Kirjeldus VARCHAR2 ( 4000 ),
     CONSTRAINT TAV_PK_Tell_seis_liik PRIMARY KEY
          (Tell_seis_liik_kood),
     CONSTRAINT TAV_TC_Tell_seis_Nim UNIQUE (Nimetus),
```

```
     CONSTRAINT TAV_TC_Tell_s_Nim CHECK (NOT
REGEXP_LIKE(Nimetus,
          '^[[:space:]]*$'))
     );

CREATE TABLE TAV_Tootja (
     Tootja_ID NUMBER ( 10 ) GENERATED AS IDENTITY NOT NULL,
     Registri_nr VARCHAR2 ( 50 ) NOT NULL,
     Nimi VARCHAR2 ( 50 ) NOT NULL,
     CONSTRAINT TAV_PK_Tootja PRIMARY KEY (Tootja_ID),
     CONSTRAINT TAV_TC_un_reg_nimi UNIQUE (Nimi, Registri_nr),
     CONSTRAINT TAV_TC_Tootja_nimi CHECK (NOT
          REGEXP_LIKE(Nimi,'^[[:space:]]*$')),
     CONSTRAINT TAV_TC_Tootja_reg CHECK (NOT
          REGEXP_LIKE(Registri_nr,'^[[:space:]]*$'))
     );

CREATE TABLE TAV_Isik (
     Isik_ID NUMBER ( 10 ) GENERATED AS IDENTITY NOT NULL,
     E_mail VARCHAR2 ( 50 ) NOT NULL,
     Parool VARCHAR2 ( 128 ) NOT NULL,
     Eesnimi VARCHAR2 ( 30 ) NOT NULL,
     Perenimi VARCHAR2 ( 30 ) NOT NULL,
     Aadress VARCHAR2 ( 100 ) NOT NULL,
     Telefon VARCHAR2 ( 20 ) NOT NULL,
     Synniaeg DATE NOT NULL,
     Sool VARCHAR2 ( 32 ) NOT NULL,
     CONSTRAINT TAV_PK_Isik PRIMARY KEY (Isik_ID),
     CONSTRAINT TAV_TC_Isik_Email UNIQUE (E_mail),
     CONSTRAINT TAV_TC_isik_pw CHECK (NOT REGEXP_LIKE(Parool,
          '^[[:space:]]*$')),
     CONSTRAINT TAV_TC_Isik_tel CHECK (NOT
          REGEXP_LIKE(Telefon,'^[[:space:]]*$') AND
          REGEXP_LIKE(Telefon, '^([[:digit:]])+$')),
     CONSTRAINT TAV_TC_Isik_Ees CHECK (REGEXP_LIKE(Eesnimi,
          '^([[:alpha:]]|[[:space:]]|[-])+$') AND NOT
          REGEXP_LIKE(Eesnimi, '^[[:space:]]*$')),
     CONSTRAINT TAV_TC_Isik_Pere CHECK (REGEXP_LIKE(Perenimi,
          '^([[:alpha:]]|[[:space:]]|[-])+$') AND NOT
          REGEXP_LIKE(Perenimi, '^[[:space:]]*$')),
     CONSTRAINT TAV_TC_isik_aadr CHECK (NOT
          REGEXP_LIKE(Aadress,'^[[:space:]]*$') AND
          REGEXP_LIKE(Aadress, '^([[:alpha:]]|[[:space:]]|[-
          ,]|[[:digit:]])+$')),
```

```
     CONSTRAINT TAV_TC_isik_email_ch CHECK (NOT
          REGEXP_LIKE(E_mail,'^[[:space:]]*$') AND
          REGEXP_LIKE(E_mail, '^([[:alpha:]]|[[:space:]]|[-
          _.@]|[[:digit:]])+$'))
     );

CREATE TABLE TAV_Ametikoht (
     Ametikoht_kood NUMBER ( 2 ) NOT NULL,
     Nimetus VARCHAR2 ( 30 ) NOT NULL,
     Kirjeldus VARCHAR2 ( 4000 ),
     CONSTRAINT TAV_PK_Ametik_kood PRIMARY KEY
(Ametikoht_kood),
     CONSTRAINT TAV_TC_Ametik_Nim UNIQUE (Nimetus),
     CONSTRAINT TAV_TC_Ak_nim CHECK (NOT
          REGEXP_LIKE(Nimetus,'^[[:space:]]*$'))
     );

CREATE TABLE TAV_Kaup (
     Kaup_ID NUMBER ( 10 ) NOT NULL,
     Tootja_ID NUMBER ( 10 ) NOT NULL,
     Kauba_kood VARCHAR2 ( 50 ) NOT NULL,
     Nimetus VARCHAR2 ( 100 ) NOT NULL,
     Hind NUMBER ( 10, 2 ) NOT NULL,
     Garantii VARCHAR2 ( 20 ) NOT NULL,
     Laoseis NUMBER ( 10 ) NOT NULL,
     CONSTRAINT TAV_TC_Kaup_Nimetus UNIQUE (Kauba_kood),
     CONSTRAINT TAV_TC_Kaup_kood UNIQUE (Nimetus),
     CONSTRAINT TAV_PK_Kaup PRIMARY KEY (Kaup_ID),
     CONSTRAINT TAV_TC_Kaup_gar CHECK (NOT
          REGEXP_LIKE(Garantii,'^[[:space:]]*$')),
     CONSTRAINT TAV_TC_kauba_kood CHECK (NOT
          REGEXP_LIKE(Kauba_kood,'^[[:space:]]*$')),
     CONSTRAINT TAV_TC_Kaup_lao CHECK (Laoseis >= 0),
     CONSTRAINT TAV_TC_Kaup_hind CHECK (Hind > 0),
     CONSTRAINT TAV_TC_Kaup_nim CHECK (NOT
          REGEXP_LIKE(Nimetus,'^[[:space:]]*$'))
     );

CREATE TABLE TAV_Tellimus (
     Tellimuse_nr NUMBER ( 10 ) GENERATED AS IDENTITY NOT NULL,
     Tell_seis_liik_kood NUMBER ( 2 ) DEFAULT 1 NOT NULL,
     Isik_ID NUMBER ( 10 ) NOT NULL,
     Esitamise_kuupaev DATE DEFAULT CURRENT_DATE NOT NULL,
     Summa NUMBER ( 10, 2 ) NOT NULL,
```

```
     Maksetahtaeg DATE DEFAULT CURRENT_DATE + 14 NOT NULL,
     CONSTRAINT TAV_PK_Tellimus PRIMARY KEY (Tellimuse_nr),
     CONSTRAINT TAV_TC_Tell_Summa CHECK (Summa > 0),
     CONSTRAINT TAV_TC_Tellimus_esitkp CHECK
          (Esitamise_kuupaev>=to_date('2010-01-01','YYYY-MM-
     DD')),
     CONSTRAINT TAV_TC_Tell_Maksekp CHECK
          (Maksetahtaeg<=to_date('2100-01-01','YYYY-MM-DD')),
     CONSTRAINT TAV_TC_Esitkp_Maksekp CHECK (Esitamise_kuupaev
<=
          Maksetahtaeg)
     );

ALTER TABLE TAV_Kaup ADD ( CONSTRAINT TAV_FK_Kaup_Tootja
FOREIGN KEY (Tootja_ID) REFERENCES TAV_Tootja (Tootja_ID));

ALTER TABLE TAV_Tootaja ADD ( CONSTRAINT TAV_FK_Toot_Isik
FOREIGN KEY (Isik_ID) REFERENCES TAV_Isik (Isik_ID) ON DELETE
CASCADE);

ALTER TABLE TAV_Tootaja ADD ( CONSTRAINT TAV_FK_Ametik_kood
FOREIGN KEY (Ametikoht_kood) REFERENCES TAV_Ametikoht
(Ametikoht_kood));

ALTER TABLE TAV_Tellimuse_rida ADD ( CONSTRAINT TAV_FK_Kaup
FOREIGN KEY (Kaup_ID) REFERENCES TAV_Kaup (Kaup_ID) ON DELETE
CASCADE);

ALTER TABLE TAV_Tellimuse_rida ADD ( CONSTRAINT
TAV_FK_Tellimus FOREIGN KEY (Tellimuse_nr) REFERENCES
TAV_Tellimus (Tellimuse_nr) ON DELETE CASCADE);

ALTER TABLE TAV_Tellimus ADD ( CONSTRAINT TAV_FK_Tellimus_Isik
FOREIGN KEY (Isik_ID) REFERENCES TAV_Klient (Isik_ID));

ALTER TABLE TAV_Tellimus ADD ( CONSTRAINT
TAV_FK_Tellimus_seis_liik FOREIGN KEY (Tell_seis_liik_kood)
REFERENCES TAV_Tellimuse_seis_liik (Tell_seis_liik_kood));

ALTER TABLE TAV_Klient ADD ( CONSTRAINT TAV_FK_Klient_Isik
FOREIGN KEY (Isik_ID) REFERENCES TAV_Isik (Isik_ID) ON DELETE
CASCADE);
```

```
ALTER TABLE TAV_Kauba_pilt ADD ( CONSTRAINT
TAV_FK_Kauba_pilt_Kaup FOREIGN KEY (Kaup_ID) REFERENCES
TAV_Kaup (Kaup_ID) ON DELETE CASCADE);

CREATE INDEX TAV_TC_Ametik_kood ON TAV_Tootaja (Ametikoht_kood
);

CREATE INDEX TAV_TC_Kaup ON TAV_Tellimuse_rida (Kaup_ID );

CREATE INDEX TAV_TC_Kaup_Tootja ON TAV_Kaup (Tootja_ID );

CREATE INDEX TAV_TC_Kaup_pilt ON TAV_Kauba_pilt (Kaup_ID );

CREATE INDEX TAV_TC_Tellimus_seis_liik ON TAV_Tellimus
(Tell_seis_liik_kood );

CREATE INDEX TAV_TC_Tellimus_Isik ON TAV_Tellimus (Isik_ID );
```

Lisa 2 - optimeeritud tabelitega andmebaasi loomise laused

```
CREATE CLUSTER OPT_Tootaja_klaster (
     Ametikoht_kood NUMBER ( 2 )
     )
     HASHKEYS 3
     SIZE 128;

CREATE CLUSTER OPT_Tell_seis_liik_klaster (
     Tell_seis_liik_kood NUMBER ( 2 )
     )
     HASHKEYS 5
     SIZE 64
     SINGLE TABLE;

CREATE TABLE OPT_Tootaja (
     Isik_ID NUMBER ( 10 ) NOT NULL,
     Ametikoht_kood NUMBER ( 2 ) NOT NULL,
     CONSTRAINT OPT_PK_Toot_Isik PRIMARY KEY (Isik_ID)
     )
     CLUSTER OPT_Tootaja_klaster (Ametikoht_kood);

CREATE TABLE OPT_Klient (
     Isik_ID NUMBER ( 10 ) NOT NULL,
     CONSTRAINT OPT_PK_Klient_Isik PRIMARY KEY (Isik_ID)
     );

CREATE TABLE OPT_Tellimuse_rida (
     Kaup_ID NUMBER ( 10 ) NOT NULL,
     Tellimuse_nr NUMBER ( 10 ) NOT NULL,
     Myygihind NUMBER ( 10, 2 ) NOT NULL,
     Kogus NUMBER ( 10 ) NOT NULL,
     CONSTRAINT OPT_PK_Tellimuse_rida PRIMARY KEY
(Tellimuse_nr,
          Kaup_ID),
     CONSTRAINT OPT_TC_Tell_r_mh CHECK (Myygihind > 0),
     CONSTRAINT OPT_TC_Tell_r_kog CHECK (Kogus > 0)
     )
     ORGANIZATION INDEX;

CREATE TABLE OPT_Kauba_pilt (
     Kauba_pilt_ID NUMBER ( 10 ) NOT NULL,
     Kaup_ID NUMBER ( 10 ) NOT NULL,
     Faili_aadress VARCHAR2 ( 50 ) NOT NULL,
```

```
     CONSTRAINT OPT_TC_Kauba_pilt UNIQUE (Faili_aadress),
     CONSTRAINT OPT_PK_Kauba_pilt PRIMARY KEY (Kauba_pilt_ID),
     CONSTRAINT OPT_TC_Kauba_pilt_Fn CHECK (NOT
          REGEXP_LIKE(Faili_aadress, '^[[:space:]]*$')),
     CONSTRAINT OPT_FK_Kauba_pilt_Kaup FOREIGN KEY (Kaup_ID)
          REFERENCES OPT_Kaup (Kaup_ID) ON DELETE CASCADE
     )
     PARTITION BY REFERENCE(OPT_FK_Kauba_pilt_Kaup);

CREATE TABLE OPT_Tellimuse_seis_liik (
     Tell_seis_liik_kood NUMBER ( 2 ) NOT NULL,
     Nimetus VARCHAR2 ( 50 ) NOT NULL,
     Kirjeldus VARCHAR2 ( 4000 ),
     CONSTRAINT OPT_PK_Tell_seis_liik PRIMARY KEY
          (Tell_seis_liik_kood),
     CONSTRAINT OPT_TC_Tell_seis_Nim UNIQUE (Nimetus),
     CONSTRAINT OPT_TC_Tell_s_Nim CHECK (NOT
REGEXP_LIKE(Nimetus,
          '^[[:space:]]*$'))
     )
     CLUSTER OPT_Tell_seis_liik_klaster (Tell_seis_liik_kood);

CREATE TABLE OPT_Tootja (
     Tootja_ID NUMBER ( 10 ) GENERATED AS IDENTITY NOT NULL,
     Registri_nr VARCHAR2 ( 50 ) NOT NULL,
     Nimi VARCHAR2 ( 50 ) NOT NULL,
     CONSTRAINT OPT_PK_Tootja PRIMARY KEY (Tootja_ID),
     CONSTRAINT OPT_TC_un_reg_nimi UNIQUE (Nimi, Registri_nr),
     CONSTRAINT OPT_TC_Tootja_nimi CHECK (NOT
          REGEXP_LIKE(Nimi,'^[[:space:]]*$')),
     CONSTRAINT OPT_TC_Tootja_reg CHECK (NOT
          REGEXP_LIKE(Registri_nr,'^[[:space:]]*$'))
     );

CREATE TABLE OPT_Isik (
     Isik_ID NUMBER ( 10 ) GENERATED AS IDENTITY NOT NULL,
     E_mail VARCHAR2 ( 50 ) NOT NULL,
     Parool VARCHAR2 ( 128 ) NOT NULL,
     Eesnimi VARCHAR2 ( 30 ) NOT NULL,
     Perenimi VARCHAR2 ( 30 ) NOT NULL,
     Aadress VARCHAR2 ( 100 ) NOT NULL,
     Telefon VARCHAR2 ( 20 ) NOT NULL,
     Synniaeg DATE NOT NULL,
     Sool VARCHAR2 ( 32 ) NOT NULL,
```

```
     CONSTRAINT OPT_PK_Isik PRIMARY KEY (Isik_ID),
     CONSTRAINT OPT_TC_Isik_Email UNIQUE (E_mail),
     CONSTRAINT OPT_TC_isik_pw CHECK (NOT REGEXP_LIKE(Parool,
          '^[[:space:]]*$')),
     CONSTRAINT OPT_TC_Isik_tel CHECK (NOT
          REGEXP_LIKE(Telefon,'^[[:space:]]*$') AND
          REGEXP_LIKE(Telefon, '^([[:digit:]])+$')),
     CONSTRAINT OPT_TC_Isik_Ees CHECK (REGEXP_LIKE(Eesnimi,
          '^([[:alpha:]]|[[:space:]]|[-])+$') AND NOT
          REGEXP_LIKE(Eesnimi, '^[[:space:]]*$')),
     CONSTRAINT OPT_TC_Isik_Pere CHECK (REGEXP_LIKE(Perenimi,
          '^([[:alpha:]]|[[:space:]]|[-])+$') AND NOT
          REGEXP_LIKE(Perenimi, '^[[:space:]]*$')),
     CONSTRAINT OPT_TC_isik_aadr CHECK (NOT
          REGEXP_LIKE(Aadress,'^[[:space:]]*$') AND
          REGEXP_LIKE(Aadress, '^([[:alpha:]]|[[:space:]]|[-
          ,]|[[:digit:]])+$')),
     CONSTRAINT OPT_TC_isik_email_ch CHECK (NOT
          REGEXP_LIKE(E_mail,'^[[:space:]]*$') AND
          REGEXP_LIKE(E_mail, '^([[:alpha:]]|[[:space:]]|[-
          _.@]|[[:digit:]])+$'))
     );

CREATE TABLE OPT_Ametikoht (
     Ametikoht_kood NUMBER ( 2 ) NOT NULL,
     Nimetus VARCHAR2 ( 30 ) NOT NULL,
     Kirjeldus VARCHAR2 ( 4000 ),
     CONSTRAINT OPT_PK_Ametik_kood PRIMARY KEY
(Ametikoht_kood),
     CONSTRAINT OPT_TC_Ametik_Nim UNIQUE (Nimetus),
     CONSTRAINT OPT_TC_Ak_nim CHECK (NOT
          REGEXP_LIKE(Nimetus,'^[[:space:]]*$'))
     )
     CLUSTER OPT_Tootaja_klaster (Ametikoht_kood);

CREATE TABLE OPT_Kaup (
     Kaup_ID NUMBER ( 10 ) NOT NULL,
     Tootja_ID NUMBER ( 10 ) NOT NULL,
     Kauba_kood VARCHAR2 ( 50 ) NOT NULL,
     Nimetus VARCHAR2 ( 100 ) NOT NULL,
     Hind NUMBER ( 10, 2 ) NOT NULL,
     Garantii VARCHAR2 ( 20 ) NOT NULL,
     Laoseis NUMBER ( 10 ) NOT NULL,
     CONSTRAINT OPT_TC_Kaup_Nimetus UNIQUE (Kauba_kood),
```

```
     CONSTRAINT OPT_TC_Kaup_kood UNIQUE (Nimetus),
     CONSTRAINT OPT_PK_Kaup PRIMARY KEY (Kaup_ID),
     CONSTRAINT OPT_TC_Kaup_gar CHECK (NOT
          REGEXP_LIKE(Garantii,'^[[:space:]]*$')),
     CONSTRAINT OPT_TC_kauba_kood CHECK (NOT
          REGEXP_LIKE(Kauba_kood,'^[[:space:]]*$')),
     CONSTRAINT OPT_TC_Kaup_lao CHECK (Laoseis >= 0),
     CONSTRAINT OPT_TC_Kaup_hind CHECK (Hind > 0),
     CONSTRAINT OPT_TC_Kaup_nim CHECK (NOT
          REGEXP_LIKE(Nimetus,'^[[:space:]]*$'))
     )
     PARTITION BY RANGE (Hind) (
          PARTITION OPT_Odav_sektsioon VALUES LESS THAN (25),
          PARTITION OPT_Alla_keskmise_sektsioon VALUES LESS
THAN
               (50),
          PARTITION OPT_Ule_keskmise_sektsioon VALUES LESS THAN
               (75),
          PARTITION OPT_Kallis_sektsioon VALUES LESS THAN
               (MAXVALUE)
     );

CREATE TABLE OPT_Tellimus (
     Tellimuse_nr NUMBER ( 10 ) GENERATED AS IDENTITY NOT NULL,
     Tell_seis_liik_kood NUMBER ( 2 ) DEFAULT 1 NOT NULL,
     Isik_ID NUMBER ( 10 ) NOT NULL,
     Esitamise_kuupaev DATE DEFAULT CURRENT_DATE NOT NULL,
     Summa NUMBER ( 10, 2 ) NOT NULL,
     Maksetahtaeg DATE DEFAULT CURRENT_DATE + 14 NOT NULL,
     CONSTRAINT OPT_PK_Tellimus PRIMARY KEY (Tellimuse_nr),
     CONSTRAINT OPT_TC_Tell_Summa CHECK (Summa > 0),
     CONSTRAINT OPT_TC_Tellimus_esitkp CHECK
          (Esitamise_kuupaev>=to_date('2010-01-01','YYYY-MM-
     DD')),
     CONSTRAINT OPT_TC_Tell_Maksekp CHECK
          (Maksetahtaeg<=to_date('2100-01-01','YYYY-MM-DD')),
     CONSTRAINT OPT_TC_Esitkp_Maksekp CHECK (Esitamise_kuupaev
<=
          Maksetahtaeg)
     )
     PARTITION BY LIST(Tell_seis_liik_kood) (
          PARTITION OPT_Esitatud_sektsioon VALUES ( 1 )
NOCOMPRESS,
```

```
          PARTITION OPT_Kohal_sektsioon VALUES ( 2 )
NOCOMPRESS,
          PARTITION OPT_Tellimisel_sektsioon VALUES ( 3 )
               NOCOMPRESS,
          PARTITION OPT_Saadetud_sektsioon VALUES ( 4 )
COMPRESS,
          PARTITION OPT_Tuhistatud_sektsioon VALUES ( 5 )
COMPRESS
     );

ALTER TABLE OPT_Kaup ADD ( CONSTRAINT OPT_FK_Kaup_Tootja
FOREIGN KEY (Tootja_ID) REFERENCES OPT_Tootja (Tootja_ID));

ALTER TABLE OPT_Tootaja ADD ( CONSTRAINT OPT_FK_Toot_Isik
FOREIGN KEY (Isik_ID) REFERENCES OPT_Isik (Isik_ID) ON DELETE
CASCADE);

ALTER TABLE OPT_Tootaja ADD ( CONSTRAINT OPT_FK_Ametik_kood
FOREIGN KEY (Ametikoht_kood) REFERENCES OPT_Ametikoht
(Ametikoht_kood));

ALTER TABLE OPT_Tellimuse_rida ADD ( CONSTRAINT OPT_FK_Kaup
FOREIGN KEY (Kaup_ID) REFERENCES OPT_Kaup (Kaup_ID) ON DELETE
CASCADE);

ALTER TABLE OPT_Tellimuse_rida ADD ( CONSTRAINT
OPT_FK_Tellimus FOREIGN KEY (Tellimuse_nr) REFERENCES
OPT_Tellimus (Tellimuse_nr) ON DELETE CASCADE);

ALTER TABLE OPT_Tellimus ADD ( CONSTRAINT OPT_FK_Tellimus_Isik
FOREIGN KEY (Isik_ID) REFERENCES OPT_Klient (Isik_ID));

ALTER TABLE OPT_Tellimus ADD ( CONSTRAINT
OPT_FK_Tellimus_seis_liik FOREIGN KEY (Tell_seis_liik_kood)
REFERENCES OPT_Tellimuse_seis_liik (Tell_seis_liik_kood));

ALTER TABLE OPT_Klient ADD ( CONSTRAINT OPT_FK_Klient_Isik
FOREIGN KEY (Isik_ID) REFERENCES OPT_Isik (Isik_ID) ON DELETE
CASCADE);

CREATE INDEX OPT_TC_Ametik_kood ON OPT_Tootaja (Ametikoht_kood
);

CREATE INDEX OPT_TC_Kaup ON OPT_Tellimuse_rida (Kaup_ID );
```

```
CREATE INDEX OPT_TC_Kaup_Tootja ON OPT_Kaup (Tootja_ID );

CREATE INDEX OPT_TC_Kaup_pilt ON OPT_Kauba_pilt (Kaup_ID )
LOCAL
 (PARTITION OPT_Odav_sektsioon,
  PARTITION OPT_Alla_keskmise_sektsioon,
  PARTITION OPT_Ule_keskmise_sektsioon,
  PARTITION OPT_Kallis_sektsioon);

CREATE INDEX OPT_TC_Tellimus_seis_liik ON OPT_Tellimus
(Tell_seis_liik_kood );

CREATE INDEX OPT_TC_Tellimus_Isik ON OPT_Tellimus (Isik_ID );
```

Lisa 3 - vaate (tellimuste nimekiri) loomise lause

```
CREATE OR REPLACE FORCE EDITIONABLE VIEW "TAV_TELL_NIMEK"
("TELLIMUSE_NR", "NIMETUS", "E_MAIL", "EESNIMI", "PERENIMI",
"SUMMA", "ESITAMISE_KUUPAEV", "MAKSETAHTAEG") AS
  select TAV_TELLIMUS.TELLIMUSE_NR as TELLIMUSE_NR,
    TAV_TELLIMUSE_SEIS_LIIK.NIMETUS as NIMETUS,
    TAV_ISIK.E_MAIL as E_MAIL,
    TAV_ISIK.EESNIMI as EESNIMI,
    TAV_ISIK.PERENIMI as PERENIMI,
    TAV_TELLIMUS.SUMMA as SUMMA,
    TAV_TELLIMUS.ESITAMISE_KUUPAEV as ESITAMISE_KUUPAEV,
    TAV_TELLIMUS.MAKSETAHTAEG as MAKSETAHTAEG
 from TAV_TELLIMUSE_SEIS_LIIK TAV_TELLIMUSE_SEIS_LIIK,
    TAV_TELLIMUS TAV_TELLIMUS,
    TAV_ISIK TAV_ISIK
 where TAV_ISIK.ISIK_ID=TAV_TELLIMUS.ISIK_ID
    and
TAV_TELLIMUSE_SEIS_LIIK.TELL_SEIS_LIIK_KOOD=TAV_TELLIMUS.TELL_
SEIS_LIIK_KOOD;
```

Lisa 4 - vaate (tellimuste detailne nimekiri) loomise lause

```
CREATE OR REPLACE FORCE EDITIONABLE VIEW "TAV_TELL_DETAIL"
("TELLIMUSE_NR", "NIMETUS", "MYYGIHIND", "KOGUS", "LAOSEIS")
AS
  select TAV_TELLIMUSE_RIDA.TELLIMUSE_NR as TELLIMUSE_NR,
    TAV_KAUP.NIMETUS as NIMETUS,
    TAV_TELLIMUSE_RIDA.MYYGIHIND as MYYGIHIND,
    TAV_TELLIMUSE_RIDA.KOGUS as KOGUS,
    TAV_KAUP.LAOSEIS as LAOSEIS
 from TAV_KAUP TAV_KAUP,
    TAV_TELLIMUSE_RIDA TAV_TELLIMUSE_RIDA,
    TAV_TELLIMUS TAV_TELLIMUS
 where TAV_TELLIMUSE_RIDA.KAUP_ID=TAV_KAUP.KAUP_ID
    and
TAV_TELLIMUSE_RIDA.TELLIMUSE_NR=TAV_TELLIMUS.TELLIMUSE_NR;
```

Lisa 5 - hetktõmmise (tellimuste nimekiri) loomise laused

```
CREATE MATERIALIZED VIEW LOG ON OPT_TELLIMUSE_SEIS_LIIK
WITH ROWID, PRIMARY KEY, SEQUENCE
INCLUDING NEW VALUES;

ALTER MATERIALIZED VIEW LOG ON OPT_TELLIMUSE_SEIS_LIIK ADD
(NIMETUS, KIRJELDUS);

CREATE MATERIALIZED VIEW LOG ON OPT_ISIK
WITH ROWID, PRIMARY KEY, SEQUENCE
INCLUDING NEW VALUES;

ALTER MATERIALIZED VIEW LOG ON OPT_ISIK ADD (E_MAIL, PAROOL,
EESNIMI, PERENIMI, AADRESS, TELEFON, SYNNIAEG, SOOL);

CREATE MATERIALIZED VIEW LOG ON OPT_TELLIMUS
WITH ROWID, PRIMARY KEY, SEQUENCE
INCLUDING NEW VALUES;

ALTER MATERIALIZED VIEW LOG ON OPT_TELLIMUS ADD
(TELL_SEIS_LIIK_KOOD, ISIK_ID, ESITAMISE_KUUPAEV, SUMMA,
MAKSETAHTAEG);

CREATE MATERIALIZED VIEW OPT_TELL_NIMEK
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
ENABLE QUERY REWRITE
AS
 select OPT_TELLIMUSE_SEIS_LIIK.ROWID as TELLIMUSE_SEIS_LIIK,
    OPT_ISIK.ROWID as ISIK,
    OPT_TELLIMUS.ROWID as TELLIMUS,
    OPT_TELLIMUS.TELLIMUSE_NR as TELLIMUSE_NR,
    OPT_TELLIMUSE_SEIS_LIIK.NIMETUS as NIMETUS,
    OPT_ISIK.E_MAIL as E_MAIL,
    OPT_ISIK.EESNIMI as EESNIMI,
    OPT_ISIK.PERENIMI as PERENIMI,
    OPT_TELLIMUS.SUMMA as SUMMA,
    OPT_TELLIMUS.ESITAMISE_KUUPAEV as KUUPAEV,
    OPT_TELLIMUS.MAKSETAHTAEG as MAKSETAHTAEG
 from OPT_TELLIMUSE_SEIS_LIIK OPT_TELLIMUSE_SEIS_LIIK,
    OPT_TELLIMUS OPT_TELLIMUS,
    OPT_ISIK OPT_ISIK
 where OPT_ISIK.ISIK_ID=OPT_TELLIMUS.ISIK_ID
```

```
    and
OPT_TELLIMUSE_SEIS_LIIK.TELL_SEIS_LIIK_KOOD=OPT_TELLIMUS.TELL_
SEIS_LIIK_KOOD;
```

Lisa 6 - hetktõmmise (tellimuste detailne nimekiri) loomise lause

```
CREATE MATERIALIZED VIEW OPT_TELL_DETAIL
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE
AS
 select TAV_TELLIMUSE_RIDA.TELLIMUSE_NR as TELLIMUSE_NR,
    TAV_KAUP.NIMETUS as NIMETUS,
    TAV_TELLIMUSE_RIDA.MYYGIHIND as MYYGIHIND,
    TAV_TELLIMUSE_RIDA.KOGUS as KOGUS,
    TAV_KAUP.LAOSEIS as LAOSEIS
 from TAV_KAUP TAV_KAUP,
    TAV_TELLIMUSE_RIDA TAV_TELLIMUSE_RIDA,
    TAV_TELLIMUS TAV_TELLIMUS
 where TAV_TELLIMUSE_RIDA.KAUP_ID=TAV_KAUP.KAUP_ID
    and
TAV_TELLIMUSE_RIDA.TELLIMUSE_NR=TAV_TELLIMUS.TELLIMUSE_NR;
```

Lisa 7 - päring andmemahtude leidmiseks

```
SELECT owner, table_name, ROUND(sum(bytes)/1024/1024, 2) MB
FROM
(SELECT segment_name table_name, owner, bytes
 FROM dba_segments
 WHERE segment_type IN ('TABLE', 'TABLE PARTITION', 'TABLE
 SUBPARTITION')
UNION ALL
SELECT i.table_name, i.owner, s.bytes
FROM dba_indexes i, dba_segments s
WHERE s.segment_name = i.index_name
AND s.owner = i.owner
AND s.segment_type IN ('INDEX', 'INDEX PARTITION', 'INDEX
SUBPARTITION')
UNION ALL
SELECT l.table_name, l.owner, s.bytes
FROM dba_lobs l, dba_segments s
WHERE s.segment_name = l.segment_name
AND s.owner = l.owner
AND s.segment_type = 'LOBSEGMENT'
UNION ALL
SELECT l.table_name, l.owner, s.bytes
FROM dba_lobs l, dba_segments s
WHERE s.segment_name = l.index_name
AND s.owner = l.owner
AND s.segment_type = 'LOBINDEX')
WHERE owner in UPPER('C##TEST') AND (table_name LIKE 'OPT_%'
OR table_name LIKE 'TAV_%')
GROUP BY table_name, owner
ORDER BY substr(table_name,1, 4), SUM(bytes) DESC;
```

Lisa 8 - rakendus tellimuse ridade genereerimiseks

```
package data_generator;

import java.io.BufferedInputStream;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

/**
 * Class for data generation.
 * @author Dan Rodionov
 */
public class Data_generator {
     /**
      * Main method.
      * @param args Arguments given to the method.
      * @throws IOException Exception given when the file could
      not         * be read.
      */
     public static void main(String[] args) throws IOException
{
          generateData();
          while (countLines("order.txt") < 1000000
                    || countLines("order.txt") > 1000100) {
               generateData();
          }
     }

     /**
      * Method for counting the number of lines.
      * @param filename File in which the lines are counted.
      * @return Number of lines in the file.
      * @throws IOException Exception given when the file could
      not         * be read.
      */
     public static int countLines(String filename)
```

```
            throws IOException {
        InputStream inputStream = new BufferedInputStream(
                new FileInputStream(filename));

        try {
            byte[] character = new byte[1024];
            int count = 0;
            int readCharacters = 0;
            boolean empty = true;
            while ((readCharacters =
                    inputStream.read(character)) != -1) {
                empty = false;
                for (int i = 0; i < readCharacters; ++i) {
                    if (character[i] == '\n') {
                        ++count;
                    }
                }
            }
            return (count == 0 && !empty) ? 1 : count;
        } finally {
            inputStream.close();
        }
    }

    /**
     * Method for generating data.
     */
    public static void generateData() {
        StringBuilder orderData = new StringBuilder();
        StringBuilder itemData = new StringBuilder();
        Random random = new Random();

        for (int i = 1; i < 100001; i++) {
            List<Integer> orderItems = new
ArrayList<Integer>();
            int items = (random.nextInt((19 - 1) + 1) + 1);
            for (int j = 0; j < items; j++) {
                orderData.append(i + "\n");
                int item = (random.nextInt((1000 - 1) + 1)
+
                    1);
                while (orderItems.contains(item)) {
                    item = (random.nextInt((1000 - 1) + 1)
                    +
```

```
                                   1);
                         }
                         itemData.append(item + "\n");
                         orderItems.add(item);
                    }
               }

               writeData("order.txt", orderData);
               writeData("item.txt", itemData);
          }

          /**
           * Method for writing data into file.
           * @param fileName File in which the data is written.
           * @param data Data being written.
           */
          public static void writeData(String fileName,
                    StringBuilder data) {
               Writer writer = null;

               try {
                    writer = new BufferedWriter(new
OutputStreamWriter(
                         new FileOutputStream(fileName), "utf-8"));
                    writer.write(data.toString());
               } catch (IOException ex) {
                    // Report.
               } finally {
                    try {
                         writer.close();
                    } catch (Exception ex) {
                    }
               }
          }
}
```