TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Mõnede ebaotstarbekate SQL päringute optimeerimine Oracle ja PostgreSQL andmebaasisüsteemides

Bakalaureusetöö

Üliõpilane: Jaan Baum

Üliõpilaskood: 112164IAPB

Juhendaja: dotsent Erki Eessaar

Tallinn
2014

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

Mõnede ebaotstarbekate SQL päringute optimeerimine Oracle ja PostgreSQL andmebaasisüsteemides

Töö eesmärk on kindlaks teha, kui palju suudavad Oracle ja PostgreSQL andmebaasisüsteemid ebaotstarbekal viisil kirjutatud SQL päringuid (SELECT lauseid) optimeerida. Töö eesmärgiks on ka võrrelda selles kitsas aspektis omavahel Oracle (Database 12c Enterprise Edition Release 1) ja PostgreSQL (9.3) andmebaasisüsteeme.

Töös luuakse Oracle ja PostgreSQL andmebaasisüsteemide abil võimalikult sarnase struktuuriga andmebaas, kuhu lisatakse genereeritud testandmed. Andmed on mõlemas andmebaasis täpselt samasugused. Seejärel käivitatakse SQL laused ja analüüsitakse lausete täitmiseks kulunud aega ning täitmisplaanide valikut.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 39 leheküljel.

Abstract

Optimization of Some Unpractical SQL Queries in Oracle and PostgreSQL Database Management Systems

The aim of this work is to find out how much can Oracle and PostgreSQL Database Management Systems (DBMSs) optimize unpractical SQL queries (SELECT statements). Another aim of this work is to compare Oracle and PostgreSQL DBMSs in this narrow field.

In this work, a similar database is created in both Oracle (Database 12c Enterprise Edition Release 1) and PostgreSQL (9.3) DBMSs and filled with generated test data. The data is the same in both databases. SQL statements are executed and the time taken and execution plans are analyzed.

The thesis is written in Estonian and contains 39 pages of text.

Sisukord

Sissejuhatus

SQL (*Structured Query Language*) on deklaratiivne keel, mille abil on võimalik kirjeldada, **mida** on andmebaasis vaja teha või milliseid andmeid leida. SQL lause põhjal koostab andmebaasisüsteem imperatiivse programmi ehk täitmisplaani, mis näitab **kuidas** soovitud tulemust saavutada. Samale SQL lausele vastavat tulemust on võimalik saavutada erinevate täitmisplaanidega. Erinevad täitmisplaanid võtavad erineva aja. Mõni täitmisplaan võtab rohkem aega, mõni vähem.

Töökiirust määravad Eessaar (2013) kohaselt näiteks:

- tabelite ühendamisel kasutatav algoritm (*nested loop join*, *hash join*, *merge join*, ...),
- indeksi(te) kasutamine,
- loetavate tabelite arv,
- tüübiteisenduste tegemine,
- funktsiooni rakendamine veeru väärtustele,
- andmete sorteerimine.

Pascal (1988, 2013) juhib tähelepanu, et SQLis on keeleline liiasus, mistõttu saab selles keeles sageli ühte ja sama ülesannet lahendada mitme erineva lausega. Pascali (1988) tehtud eksperiment ühe lihtsa andmete otsimise ülesandega näitas, et erinevate lahenduste puhul võib andmebaasisüsteem kasutada erinevaid täitmisplaane, mistõttu võtab soovitud tulemuseni jõudmine erineva hulga aega. Erinevates andmebaasisüsteemides võib lausete töökiiruse pingerida olla erinev. Fernandez (2011) demonstreerib, et selline probleem on ka uuemate andmebaasisüsteemidega.

Selle töö eesmärgiks on uurida mõnede ebaotstarbekal viisil kirjutatud SQL päringute (SELECT lausete) täitmisplaane kahe andmebaasisüsteemi näitel. Selle töö kontekstis on SQL lause ebaotstarbekas kui on võimalik kirjutada mõni teine SQL lause, mis lahendab sama ülesande kiiremini kuid muus osas samasuguste tulemustega. Sageli on need ebaotstarbekad laused ka pikemad ning inimkasutajale raskemini mõistetavamad kui parema töökiirusega laused.

Andmebaasisüsteemidena kasutan Oracle Database 12c Enterprise Edition Release 1 (Oracle, 2014) ja PostgreSQL 9.3 (PostgreSQL, 2014). Andmebaasisüsteemidega

töötamiseks kasutan käsureapõhiseid programme SQL Plus (Oracle) ja psql (PostgreSQL) ning veebirakendusi Oracle Application Express ja phpPgAdmin. Töös kasutatud andmebaasisüsteemid olid ette antud ülesande püstituses. Autor on nende andmebaasisüsteemidega kokku puutunud ning need kuuluvad maailma populaarseimate andmebaasisüsteemide hulka (DB-Engines Ranking, 2014).

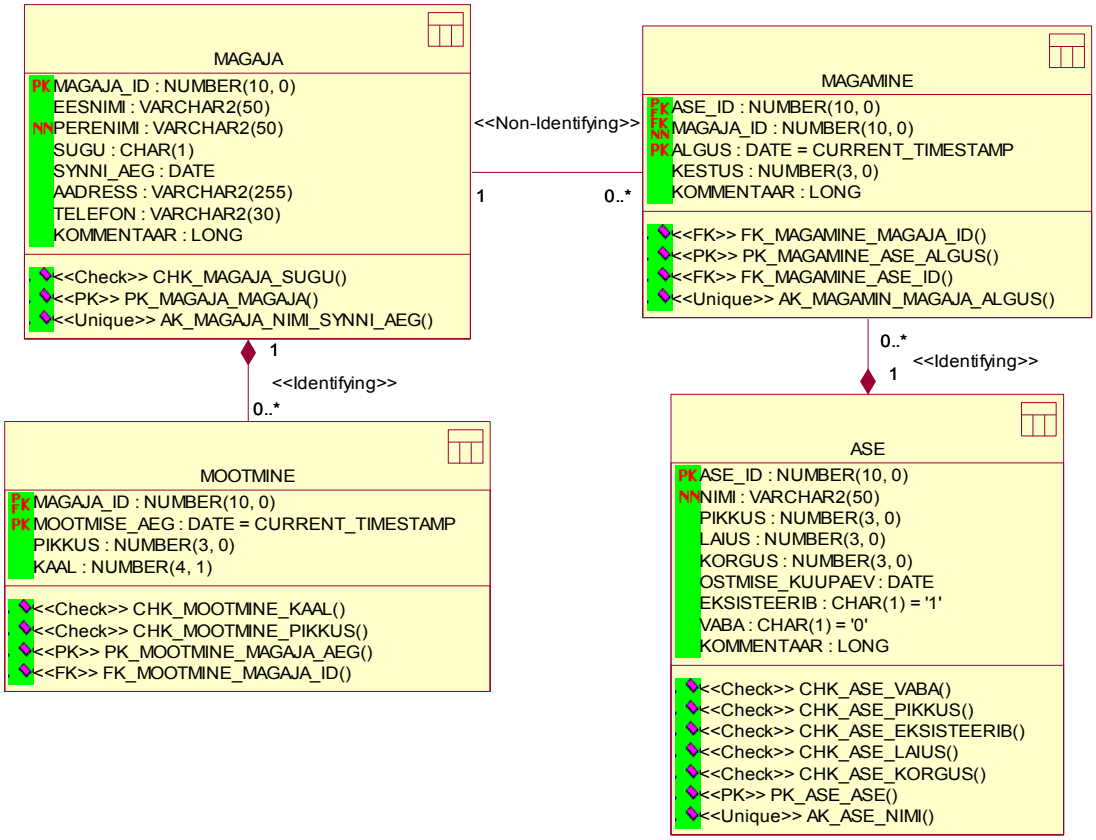
Kuna andmebaasisüsteemid ja andmebaasid asuvad samas serverarvutis (apex.ttu.ee, CPU: 15 x 2200 MHz x86_64, muutmälu: 40 GB), siis võimaldab see antud töö kitsas aspektis võrrelda ka Oracle ja PostgreSQL töökiirust samasuguste ülesannete lahendamisel ühesugustes oludes.

Töös kasutatud magajate andmebaasi struktuur ja SQL laused on antud juhendaja poolt (Eessaar, 2014). Neid kasutatakse õppeaine "Andmebaasid I" õppematerjalides ja need illustreerivad SQL kontrolltööde vastustes esinevaid levinud probleeme.

Töös luuakse andmebaasidesse tabelid, genereeritakse testandmed INSERT lausetena ja sisestatakse andmed andmebaasidesse. Seejärel käivitatakse SQL laused ja tuuakse välja täitmiseks kulunud aeg ning täitmisplaan.

1. Andmebaas

Andmebaasis registreeritakse andmed asemete, magajate, magamiste ja mõõtmiste kohta. Magamisega on seotud ase ja magaja (magaja magab asemel). Mõõtmisega on seotud magaja, keda mõõdetakse. Joonisel 1 esitatakse andmebaasi struktuuri kirjeldav diagramm (loodud Rational Rose tarkvaraga, kasutades selle pöördprojekteerimise funktsionaalsust), kus kasutatakse Oracle andmebaasis tarvitatavaid andmetüüpe.



Joonis 1: Oracle andmebaasi tabelite struktuur

1.1 Tabelite detailsed kirjeldused

Järgnevalt on esitatud tabelite *Ase* (vt Tabel 1), *Magaja* (vt Tabel 2), *Magamine* (vt Tabel 3) ja *Mootmine* (vt Tabel 4) veerud ja nende andmetüüpide võrdlus töös kasutatavates andmebaasisüsteemides.

Tabel 1: Tabeli Ase veergude nimed ja andmetüübid

Ase		
Veeru nimi	**Oracle**	**PostgreSQL**
ase_id	NUMBER(10,0)	integer
nimi	VARCHAR2(50 BYTE)	character varying(50)
pikkus	NUMBER(3,0)	smallint
laius	NUMBER(3,0)	smallint
korgus	NUMBER(3,0)	smallint
ostmise_kuupaev	DATE	date
eksisteerib	CHAR(1 BYTE)	boolean
vaba	CHAR(1 BYTE)	boolean
kommentaar	LONG	text

Tabel 2: Tabeli Magaja veergude nimed ja andmetüübid

Magaja		
Veeru nimi	**Oracle**	**PostgreSQL**
magaja_id	NUMBER(10,0)	integer
eesnimi	VARCHAR2(50 BYTE)	character varying(50)
perenimi	VARCHAR2(50 BYTE)	character varying(50)
sugu	CHAR(1 BYTE)	character(1)
synni_aeg	DATE	date
aadress	VARCHAR2(255 BYTE)	character varying(255)
telefon	VARCHAR2(30 BYTE)	character varying(30)
kommentaar	LONG	text

Tabel 3: Tabeli Magamine veergude nimed ja andmetüübid

Magamine		
Veeru nimi	**Oracle**	**PostgreSQL**
ase_id	NUMBER(10,0)	integer
magaja_id	NUMBER(10,0)	integer
algus	DATE	timestamp without time zone
kestus	NUMBER(3,0)	smallint
kommentaar	LONG	text

Tabel 4: Tabeli Mootmine veergude nimed ja andmetüübid

Mootmine		
Veeru nimi	**Oracle**	**PostgreSQL**
magaja_id	NUMBER(10,0)	integer
mootmise_aeg	DATE	timestamp without time zone
pikkus	NUMBER(3,0)	smallint
kaal	NUMBER(4,1)	numeric(4,1)

Tabelitele on automaatselt süsteemi poolt loodud indeksid primaarvõtme (PRIMARY KEY) ja unikaalsuse (UNIQUE) kitsenduste tõttu (vt Tabel 5).

Tabel 5: Tabelitele loodud indeksid

Tabel	Indeksi nimi	Veerg/Veerud
Ase	pk_ase_ase	ase_id
	ak_ase_nimi	nimi
Magaja	pk_magaja_magaja	magaja_id
	ak_magaja_nimi_synni_aeg	eesnimi, perenimi, synni_aeg
Magamine	pk_magamine_ase_algus	ase_id, algus
	ak_magamine_magaja_algus	magaja_id, algus
Mootmine	pk_mootmine_magaja_aeg	magaja_id, mootmise_aeg

Kõikidele välisvõtme veergudele on loodud välisvõtme kitsendus (FOREIGN KEY). Üldiselt soovitatakse välisvõtmed indekseerida. Vaadates Tabelit 5 on näha, et automaatselt

loodud indeksid katavad ka kõik välisvõtmed.

Tabeli *Magaja* veerule sugu on loodud CHECK kitsendus, mis lubab väärtusena kasutada ainult tähemärke 'M' või 'N'. Tabeli *Ase* veergudele *pikkus, laius* ja *korgus* ning tabeli *Mootmine* veergudele *kaal* ja *pikkus* on loodud CHECK kitsendused, mis piiravad väärtuste vahemikku.

Oracle puhul on tõeväärtuse tüüpi andmete salvestamiseks kasutatud andmetüüpi CHAR(1 BYTE), milles on CHECK kitsenduse abil lubatud ainult väärtused '0' või '1'.

Loodud on ka kaks arvujada generaatorit: üks annab täisarvulisi väärtuseid tabeli *Ase* veerule *ase_id* ja teine annab täisarvulisi väärtuseid tabeli *Magaja* veerule *magaja_id*.

1.2 Andmed

Testandmete genereerimiseks kasutan enda loodud Java programmi, mis genereerib INSERT laused kõigi nelja tabeli jaoks. Programmi kood on lisas Lisa 1. Kasutan täpselt samasuguseid INSERT lauseid mõlema andmebaasisüsteemi täitmiseks andmetega, ehk laused on viidud kujule, mis sobivad mõlema süsteemi jaoks. Järgnevalt on toodud mõned näited genereeritud INSERT lausetest.

```
INSERT INTO ase (nimi,pikkus,laius,korgus,ostmise_kuupaev,
eksisteerib,vaba,kommentaar) VALUES ('t1vPJAaWrB3H',175,
108,14,'1960-3-19','0','1','GXYcuq7GxmiPa3YfqndPt3FtLgHl
plOcZ');

INSERT INTO magaja (eesnimi,perenimi,sugu,synni_aeg,aadress,
telefon,kommentaar) VALUES ('qmSsOtsT','Ug8vImDlwg
A8N9xjHrb2Svpu','M','1949-5-18','SX3 UCnhN s5BRaOrkcBM ',
'9rUva4lBb9fiUz7R','pEKz0');

INSERT INTO magamine (ase_id,magaja_id,algus,kestus,
kommentaar) VALUES (7567,88,'1959-1-24 17:31',721,'oGKYLD');
```

```
INSERT INTO mootmine (magaja_id,mootmise_aeg,pikkus,kaal)
VALUES (4285,'1997-9-18 3:23',204,99.22154);
```

Tekstitüüpi andmed genereeritakse juhusliku pikkusega, lubatud piirides ja sisaldavad juhuslikus järjestuses suurtähti (A-Z), väiketähti (a-z), numbreid (0-9) ja tühikuid.

Arvutüüpi andmed genereeritakse juhuslikult, lubatud piirides.

Kuupäevatüüpi andmed (*date*) genereeritakse vahemikus '1940-1-1' kuni '2000-12-28'. Ajatempli tüüpi andmetele (*timestamp*) lisatakse ka kellaaeg vahemikus '0:0' kuni '23:59'.

Tõeväärtustüüpi väljade väärtuseks võetakse juhuslikult '0' või '1'. PostgreSQL teisendab selle *boolean* tüüpi väärtuseks.

Tabeli *Magaja* veerule sugu valitakse juhuslikult 'M' või 'N'.

Kõigis tabelites on 100 000 rida.

1.3 Statistika

Pärast andmete sisestamist ja enne päringute käivitamist värskendan tabelite ja indeksite kohta käivat statistikat, sest see võimaldab andmebaasisüsteemil valida antud kontekstis kõige optimaalsema täitmisplaani. Oracle puhul kasutan Application Express veebirakendust, kus valin iga tabeli kohta Statistics -> Analyze -> Compute Statistics ja PostgreSQL puhul käivitan iga tabeli jaoks SQL lause: `ANALYZE [tabeli nimi]`.

2. Uuritavad SQL laused ja uuringu tulemused

Järgnevalt on esitatud ülesanded ja neid lahendavad SQL laused. Iga SQL lause juures on ära toodud selle täitmise aeg ja täitmisplaani sõnaline kirjeldus mõlemas andmebaasisüsteemis.

Täitmisplaani ja täitmise aja nägemiseks ...

... Oracle puhul seadistan sessiooni lausetega:

```
SET AUTOTRACE TRACEONLY
SET TIMING ON
```

... PostgreSQL puhul kasutan iga lause ees `EXPLAIN ANALYZE`.

Juhul, kui lause täitmine võtab palju aega (rohkem kui 10 minutit), siis vaatan ainult täitmisplaani, ilma lause täitmiseta. Oracle puhul kasutan lause ees `EXPLAIN PLAN FOR`. PostgreSQL puhul kasutan lause ees ainult `EXPLAIN`.

Iga ebaotstarbeka lahenduse näite korral tuuakse välja kaks või rohkem lauset. Lausete juures kasutatakse tähistust:

- (E) – mingis aspektis ebaotstarbekas lahendus

- (O) – otstarbekam lahendus.

Ebaotstarbekate lausete tekstis on üleliigsed/ebaotstarbekad kohad tähistatud **rasvaselt**.

Lausete täitmise aega mõõdetakse millisekundites (ms). Igat lauset on käivitatud vähemalt 3 korda ja leitud keskmine täitmise aeg.

2.1 Skalaarse funktsiooni väljakutse

Ülesanne: Leia hetke kuupäev.

Lause 1 (E)

```
SELECT DISTINCT current_date AS kp FROM Ase;
```

Oracle

Täitmise aeg: 60 ms

Täitmisplaan: Süsteem kasutab tabelile *Ase* loodud indeksit `pk_ase_ase` (*INDEX FAST FULL SCAN*) ja leiab kuupäeva kõikide tabelis olevate ridade kohta ning seejärel eemaldab korduvad read (*HASH UNIQUE*).

PostgreSQL

Täitmise aeg: 126 ms

Täitmisplaan: Süsteem leiab hetke kuupäeva kõikide tabelis olevate ridade kohta kasutades tabeli *Ase* täielikku läbiskaneerimist ning seejärel eemaldab korduvad read.

Lause 2 (O)

Oracle

```
SELECT current_date AS kp FROM dual;
```

Täitmise aeg: 10 ms

Täitmisplaan: Süsteem leiab hetke kuupäeva. Süsteemses abitabelis *dual* on ainult üks rida.

PostgreSQL

```
SELECT current_date AS kp;
```

Täitmise aeg: 0 ms

Täitmisplaan: Süsteem leiab hetke kuupäeva.

Kommentaar: Esimese lause puhul loevad mõlemad andmebaasisüsteemid tabelit *Ase* või sellega seotud indeksit ja eemaldavad korduvad read, kuigi tulemuse jaoks pole seda vaja teha. Sisuliselt leitakse hetke kuupäev 100 000 korda (just nii palju ridu on tabelis *Ase)* ja siis jäetakse leitud väärtustest alles ainult üks. Seetõttu on teine lause kiirem.

2.2 Korduste eemaldamine

Ülesanne: Leia magajate numbrilised identifikaatorid ja perenimed.

Lause 1 (E)

```
SELECT DISTINCT magaja_id, perenimi FROM Magaja;
```

Oracle

Täitmise aeg: 1120 ms

Täitmisplaan: Süsteem leiab tabeli *Magaja* täieliku läbiskaneerimisega kõik read.

PostgreSQL

Täitmise aeg: 520 ms

Täitmisplaan: Süsteem leiab tabeli *Magaja* täieliku läbiskaneerimisega kõik read ja eemaldab korduvad read kasutades sorteerimist (*external sort*).

Lause 2 (E)

```
SELECT magaja_id, perenimi FROM Magaja GROUP BY magaja_id, perenimi;
```

Oracle

Täitmise aeg: 1180 ms

Täitmisplaan: Süsteem leiab tabeli *Magaja* täieliku läbiskaneerimisega kõik read ja grupeerib tulemuse (*HASH GROUP BY*).

PostgreSQL

Täitmise aeg: 501 ms

Täitmisplaan: Süsteem leiab tabeli Magaja täieliku läbiskaneerimisega kõik read. Seejärel sorteeritakse ja grupeeritakse tulemus (*external sort*).

Lause 3 (O)

```
SELECT magaja_id, perenimi FROM Magaja;
```

Oracle

Täitmise aeg: 850 ms

Täitmisplaan: Süsteem leiab tabeli Magaja täieliku läbiskaneerimisega kõik read.

PostgreSQL

Täitmise aeg: 90 ms

Täitmisplaan: Süsteem leiab tabeli Magaja täieliku läbiskaneerimisega kõik read.

Kommentaar: Oracle kasutab esimese ja kolmanda lause puhul sama täitmisplaani ehk primaarvõtme unikaalsuse tõttu ei hakka esimese lause puhul korduvaid ridu eemaldama. PostgreSQL seda lihtsustust ei tee, kuid ebaotstarbekate lausete täitmine võtab siiski vähem aega kui Oracles.

2.3 Läbimõtlemata tingimused

Ülesanne: Leia üle 298 cm pikad asemed.

Tabeli *Ase* veerul *pikkus* ei ole esialgu indeksit.

Lause 1 (E)

```
SELECT * FROM Ase WHERE pikkus > '298';
```

Oracle

Täitmise aeg: 120 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`.

PostgreSQL

Täitmise aeg: 40 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`.

Lause 2 (E)

```
SELECT * FROM Ase WHERE ase_id IN (SELECT ase_id FROM Ase
WHERE pikkus > 298);
```

Oracle

Täitmise aeg: 120 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`.

PostgreSQL

Täitmise aeg: 50 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`. Vahetulemus ühendatakse tabeliga Ase (*Nested Loop*), kasutades indeksit `pk_ase_ase` (*Index Scan*).

Lause 3 (E)

```
SELECT * FROM Ase WHERE pikkus IN (SELECT pikkus FROM Ase
WHERE pikkus > 298);
```

Oracle

Täitmise aeg: 150 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`. Vahetulemus ühendatakse tabeliga Ase (*HASH JOIN RIGHT SEMI*).

PostgreSQL

Täitmise aeg: 166 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Ase* täieliku läbiskaneerimise ja ühendab tulemused omavahel (*Hash Join*).

Lause 4 (O)

```
SELECT * FROM Ase WHERE pikkus > 298;
```

Oracle

Täitmise aeg: 120 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`.

PostgreSQL

Täitmise aeg: 38 ms

Täitmisplaan: Süsteem leiab tabeli *Ase* täieliku läbiskaneerimisega kõik read, kus on täidetud tingimus `pikkus > 298`.

Kommentaar: Oracle kasutab esimese, teise ja neljanda lause puhul sama täitmisplaani, kuid PostgreSQL täitmise aeg on nende lausete puhul väiksem.

Lisan tabeli *Ase* veerule *pikkus* indeksi ning värskendan tabeli statistikat. Seejärel proovin uuesti kahte lauset. Lause 1 korral on andmebaasisüsteem sunnitud tegema sisemisi tüübiteisendusi, mis peaks välistama indeksi kasutamise.

Lause 1 (E)

```
SELECT * FROM Ase WHERE pikkus > '298';
```

Oracle

Täitmise aeg: 70 ms

Täitmisplaan: Süsteem kasutab tabeli *Ase* veerule *pikkus* loodud indeksit (*INDEX RANGE SCAN*) ja leiab read, mis vastavad tingimusele `pikkus > 298`.

PostgreSQL

Täitmise aeg: 2 ms

Täitmisplaan: Süsteem kasutab tabeli *Ase* veerule *pikkus* loodud indeksit (*Bitmap Index Scan*), et leida tingimusele `pikkus > 298` vastavate ridade füüsilised asukohad. Seejärel loetakse ridade andmed (*Bitmap Heap Scan*).

Lause 4 (O)

```
SELECT * FROM Ase WHERE pikkus > 298;
```

Oracle

Täitmise aeg: 70 ms

Täitmisplaan: Süsteem kasutab tabeli *Ase* veerule *pikkus* loodud indeksit (*INDEX RANGE SCAN*) ja leiab read, mis vastavad tingimusele `pikkus > 298`.

PostgreSQL

Täitmise aeg: 2 ms

Täitmisplaan: Süsteem kasutab tabeli *Ase* veerule *pikkus* loodud indeksit (*Bitmap Index Scan*), et leida tingimusele `pikkus > 298` vastavate ridade füüsilised asukohad. Seejärel loetakse ridade andmed (*Bitmap Heap Scan*).

Kommentaar: Oracle kasutab mõlema lause täitmiseks sama täitmisplaani ja PostgreSQL kasutab mõlema lause täitmiseks sama täitmisplaani. Mõlema lause puhul kasutatakse indeksit.

2.4 Liigsetest tabelitest andmete küsimine

Ülesanne: Leia selliste magajate numbrilised identifikaatorid, kellel on vähemalt üks seotud magamine.

Lause 1 (E)

```
SELECT DISTINCT Magaja.magaja_id FROM Magaja INNER JOIN Magamine ON Magaja.magaja_id = Magamine.magaja_id;
```

Oracle

Täitmise aeg: 570 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*), et leida `magaja_id` väärtused. Seejärel eemaldatakse korduvad read (*HASH UNIQUE*).

PostgreSQL

Täitmise aeg: 589 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*) ja tabelile *Magaja* loodud indeksit `pk_magaja_magaja` (*Index Only Scan*), et ühendada tabelid (*Merge Join*). Seejärel eemaldatakse korduvad read (*Unique*).

Lause 2 (O)

```
SELECT DISTINCT magaja_id FROM Magamine WHERE magaja_id IS NOT NULL;
```

Oracle

Täitmise aeg: 550 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*), et leida `magaja_id` väärtused. Seejärel eemaldatakse korduvad read (*HASH UNIQUE*).

PostgreSQL

Täitmise aeg: 284 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*), et leida kõik read, kus tingimus `magaja_id IS NOT NULL` on tõene ja seejärel eemaldab korduvad read (*Unique*).

Kommentaar: Oracle valib mõlema lause täitmiseks sama täitmisplaani, milles ei loeta andmeid tabelist *Magaja* ega selle indeksitest (toimub tabeli elimineerimise teisendus). PostgreSQL ei osanud ebaotstarbeka lahenduse korral täitmisplaanist tabeli *Magaja* kasutamist elimineerida.

2.5 Liigsetest tabelitest andmete küsimine (2)

Ülesanne: Leia isikud, kellega on seotud vähemalt ühe magamise andmed. Väljasta andmed kõigist tabeli *Magaja* veergudest.

Lause 1 (E)

```
SELECT * FROM Magaja WHERE magaja_id IN
(SELECT Magaja.magaja_id FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id = Magamine.magaja_id);
```

Oracle

Täitmise aeg: 9470 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite poolühendamine (*HASH JOIN RIGHT SEMI*).

PostgreSQL

Täitmise aeg: 827 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*) ja tabelile *Magaja* loodud indeksit `pk_magaja_magaja` (*Index Only Scan*), et läbi viia tabelite ühendamine (*Merge Join*). Vahetulemus materialiseeritakse. Süsteem kasutab tabelile *Magaja* loodud indeksit `pk_magaja_magaja` (*Index Scan*), et läbi viia poolühendamine vahetulemusega (*Merge Semi Join*).

Lause 2 (O)

```
SELECT * FROM Magaja WHERE magaja_id IN
(SELECT magaja_id FROM Magamine WHERE magaja_id IS NOT
NULL);
```

Oracle

Täitmise aeg: 9450 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite poolühendamine (*HASH JOIN RIGHT SEMI*).

PostgreSQL

Täitmise aeg: 525 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*), et leida kõik read, kus tingimus `magaja_id IS NOT NULL` on tõene ja tabelile *Magaja* loodud indeksit `pk_magaja_magaja` (*Index Scan*), et läbi viia tabelite poolühendamine (*Merge Semi Join*).

Kommentaar: Oracle valib mõlema lause täitmiseks sama täitmisplaani, kuid PostgreSQL on nii ebaotstarbeka kui otstarbeka lause täitmisel kordades kiirem.

2.6 Probleemid grupeerimisega

Ülesanne: Leia iga magaja kohta, kellel on vähemalt üks seotud magamine, temaga seotud magamiste arv. Väljasta magaja numbriline identifikaator, perenimi ja magamiste arv.

Lause 1 (E)

```
SELECT M.magaja_id, M.perenimi, Count(*) AS arv
FROM Magaja M, Magamine Ma
GROUP BY M.magaja_id, M.perenimi, Ma.magaja_id
HAVING M.magaja_id = Ma.magaja_id;
```

Oracle

Täitmise aeg: rohkem kui 10 min

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja sorteerimist (*BUFFER SORT*), et läbi viia otsekorrutis tabeliga *Magaja* (*MERGE JOIN CARTESIAN*). Seejärel grupeeritakse tulemus (*HASH GROUP BY*) ning leitakse read, kus on täidetud tingimus `M.magaja_id = Ma.magaja_id`.

PostgreSQL

Täitmise aeg: 1050 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*).

Lause 1 kommentaar: PostgreSQL teisendab otsekorrutise tabelite ühendamiseks ja seetõttu suudab tulemuse mõistliku ajaga väljastada. Oracle seda ei tee ja seetõttu võtab tulemuse väljastamine väga kaua aega, sest kõigepealt leitakse otsekorrutis (100 000 * 100 000 rida), mida hakatakse piirama.

Lause 2 (E)

```
SELECT M.magaja_id, M.perenimi, Count(*) AS arv
FROM Magaja M, Magamine Ma WHERE M.magaja_id = Ma.magaja_id
GROUP BY M.magaja_id, M.perenimi HAVING Count(*)>=1;
```

Oracle

Täitmise aeg: 1030 ms

Täitmisplaan: Süsteem kasutab tabelile Magamine loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli Magaja täielikku läbiskaneerimist, et läbi viia tabelite ühendamine (*HASH JOIN*). Seejärel grupeeritakse tulemus (*HASH GROUP BY*) ning leitakse read, kus on täidetud tingimus `Count(*)>=1`.

PostgreSQL

Täitmise aeg: 898 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*) ning leitakse read, kus on täidetud tingimus `Count(*)>=1`.

Lause 3 (O)

```
SELECT M.magaja_id, M.perenimi, Count(*) AS arv
FROM Magaja M, Magamine Ma WHERE M.magaja_id = Ma.magaja_id
GROUP BY M.magaja_id, M.perenimi;
```

Oracle

Täitmise aeg: 1010 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) grupeerimiseks (*HASH GROUP BY*). Vahetulemus materialiseeritakse. Süsteem kasutab tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia ühendamine vahetulemusega (*HASH JOIN*). Tulemus grupeeritakse (*HASH GROUP BY*).

PostgreSQL

Täitmise aeg: 1186 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*).

Kommentaar: PostgreSQL kasutab esimese ja kolmanda lause puhul sama täitmisplaani ehk teisendab otsekorrutise tabelite ühendamiseks.

2.7 LEFT JOIN vs. INNER JOIN

Ülesanne: Leia isikud, kellega on seotud vähemalt kahe magamise andmed tabelis Magamine.

Lause 1 (E)

```
SELECT Magaja.magaja_id, perenimi FROM Magaja
LEFT JOIN Magamine ON Magaja.magaja_id = Magamine.magaja_id
GROUP BY Magaja.magaja_id, perenimi HAVING Count(*)>=2;
```

Oracle

Täitmise aeg: 620 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite välisühendamine (*HASH JOIN RIGHT OUTER*). Seejärel grupeeritakse tulemus (*HASH GROUP BY*) ning leitakse read, kus on täidetud tingimus `Count(*)>=2`.

PostgreSQL

Täitmise aeg: 1155 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Right Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*) ning leitakse read, kus on täidetud tingimus `Count(*)>=2`.

Lause 2 (O)

```
SELECT Magaja.magaja_id, perenimi FROM Magaja
INNER JOIN Magamine ON Magaja.magaja_id = Magamine.magaja_id
GROUP BY Magaja.magaja_id, perenimi HAVING Count(*)>=2;
```

Oracle

Täitmise aeg: 360 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite ühendamine (*HASH JOIN*). Seejärel grupeeritakse tulemus (*HASH GROUP BY*) ning leitakse read, kus on täidetud tingimus `Count(*)>=2`.

PostgreSQL

Täitmise aeg: 1020 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*) ning leitakse read, kus on täidetud tingimus `Count(*)>=2`.

Kommentaar: Välisühendamise operatsioon `LEFT JOIN` võtab rohkem aega kui `INNER JOIN` operatsioon.

2.8 Korduste eemaldamine (2)

Ülesanne: Leia inimeste erinevad eesnimed. Ära arvesta eesnime puudumist eraldi eesnimena.

Lause 1 (E)

```
SELECT DISTINCT eesnimi FROM Magaja WHERE eesnimi IS NOT NULL GROUP BY eesnimi;
```

Oracle

Täitmise aeg: 950 ms

Täitmisplaan: Süsteem kasutab tabelile *Magaja* loodud indeksit `ak_magaja_nimi_synni_aeg` (*INDEX FAST FULL SCAN*), et leida kõik read, kus on täidetud tingimus `eesnimi IS NOT NULL`. Seejärel grupeeritakse tulemus (*HASH GROUP BY*).

PostgreSQL

Täitmise aeg: 1454 ms

Täitmisplaan: Süsteem leiab tabeli *Magaja* täielikku läbiskaneerimist kasutades kõik read, kus on täidetud tingimus `eesnimi IS NOT NULL`. Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*) ning eemaldatakse korduvad read (*Unique*).

Lause 2 (O)

```
SELECT DISTINCT eesnimi FROM Magaja WHERE eesnimi IS NOT NULL;
```

Oracle

Täitmise aeg: 990 ms

Täitmisplaan: Süsteem kasutab tabelile *Magaja* loodud indeksit `ak_magaja_nimi_synni_aeg` (*INDEX FAST FULL SCAN*), et leida kõik read, kus on täidetud tingimus `eesnimi IS NOT NULL`. Seejärel eemaldatakse korduvad read (*HASH UNIQUE*).

PostgreSQL

Täitmise aeg: 1163 ms

Täitmisplaan: Süsteem leiab tabeli *Magaja* täielikku läbiskaneerimist kasutades kõik read, kus on täidetud tingimus `eesnimi IS NOT NULL`. Seejärel sorteeritakse tulemus (*external merge*) ning eemaldatakse korduvad read (*Unique*).

Kommentaar: Oracle tulemus on kiirem, sest Oracle kas ainult grupeerib või eemaldab korduvad read, kuid PostgreSQL teeb esimese lause puhul mõlemat. Samuti kasutab Oracle indeksit, kuid PostgreSQL eelistab tabeli täielikku läbiskaneerimist.

2.9 Korduste eemaldamine (3)

Ülesanne: Leia iga vähemalt ühe magamisega seotud isiku kohta, temaga seotud magamiste arv.

Lause 1 (E)

```
SELECT DISTINCT Magaja.magaja_id, perenimi, Count(*) AS arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, perenimi;
```

Oracle

Täitmise aeg: 860 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) grupeerimiseks (*HASH GROUP BY*). Vahetulemus materialiseeritakse. Süsteem kasutab tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia ühendamine vahetulemusega (*HASH JOIN*). Tulemus grupeeritakse (*HASH GROUP BY*).

PostgreSQL

Täitmise aeg: 1298 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*). Saadud tulemus sorteeritakse (*external merge*) ning eemaldatakse korduvad read (*Unique*).

Lause 2 (O)

```
SELECT Magaja.magaja_id, perenimi, Count(*) AS arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, perenimi;
```

Oracle

Täitmise aeg: 890 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) grupeerimiseks (*HASH GROUP BY*). Vahetulemus materialiseeritakse. Süsteem kasutab tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia ühendamine vahetulemusega (*HASH JOIN*). Tulemus grupeeritakse (*HASH GROUP BY*).

PostgreSQL

Täitmise aeg: 922 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (Hash Join). Seejärel sorteeritakse ja grupeeritakse tulemus (external merge).

Kommentaar: Oracle valib mõlema lause täitmiseks sama täitmisplaani. Oracle tulemus on kiirem, sest Oracle ainult grupeerib, kuid PostgreSQL sorteerib esimese lause puhul teist korda, et korduvaid ridu eemaldada.

2.10 Tingimused, mis välistavad indeksi kasutamise

Ülesanne: Leia andmed magamiste kohta, mille aseme identifikaator on vahemikus 900 kuni 2500 (otspunktid kaasa arvatud) ning mis on alanud vähem kui 10 000 päeva tagasi.

Lause 1 (E)

Oracle

```
SELECT * FROM Magamine WHERE ase_id BETWEEN 900 AND 2500 AND
algus+10000>current_date;
```

Täitmise aeg: 170 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `pk_magamine_ase_algus` (*INDEX RANGE SCAN*), et leida tingimusele `ase_id>=900 AND ase_id<=2500` vastavad read. Nende hulgast leitakse tingimusele `algus+10000>current_date` vastavad read.

PostgreSQL

```
SELECT * FROM Magamine WHERE ase_id BETWEEN 900 AND 2500 AND
algus+interval '10000 days'>current_date;
```

Täitmise aeg: 18 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `pk_magamine_ase_algus` (*Bitmap Index Scan*), et leida tingimusele `ase_id>=900 AND ase_id<=2500` vastavate ridade füüsilised asukohad. Seejärel loetakse ridade andmed (*Bitmap Heap Scan*) ja leitakse read, mis vastavad tingimusele `algus+interval '10000 days'>current_date`.

Lause 2 (O)

Oracle

```
SELECT * FROM Magamine WHERE ase_id BETWEEN 900 AND 2500 AND
algus>current_date-10000;
```

Täitmise aeg: 130 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `pk_magamine_ase_algus` (*INDEX RANGE SCAN*), et leida tingimusele `ase_id>=900 AND algus>current_date-10000 AND ase_id<=2500` vastavad read. Nende hulgast leitakse tingimusele `algus>current_date-10000` vastavad read.

PostgreSQL

```
SELECT * FROM Magamine WHERE ase_id BETWEEN 900 AND 2500 AND
algus>current_date-interval '10000 days';
```

Täitmise aeg: 6 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `pk_magamine_ase_algus` (*Bitmap Index Scan*), et leida tingimusele `ase_id>=900 AND ase_id<=2500 AND algus>current_date-interval '10000 days'` vastavate ridade füüsilised asukohad. Seejärel loetakse ridade andmed (*Bitmap Heap Scan*).

Kommentaar: Ebaotstarbeka lause täitmine (funktsiooni rakendamine veeru väärtustele) võtab natuke rohkem aega kui teise lause täitmine. Mõlemad andmebaasisüsteemid kasutavad mõlema lause puhul indeksit, et leida read, mille `ase_id` on õiges vahemikus.

2.11 Poolühendamine

Ülesanne: Leia kõik magajad, kellega on seotud vähemalt üks magamine. Väljasta andmed kõigist tabeli *Magaja* veergudest.

Lause 1 (E)

```
SELECT * FROM Magaja WHERE (SELECT Count(*) AS arv FROM Magamine
WHERE Magamine.magaja_id = Magaja.magaja_id)>0;
```

Oracle

Täitmise aeg: 9890 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite poolühendamine (*HASH JOIN RIGHT SEMI*).

PostgreSQL

Täitmise aeg: 1084 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise. Iga rea kohta kasutatakse tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*), et leida ja loendada ridu, kus on täidetud tingimus `Magamine.magaja_id = Magaja.magaja_id`.

Lause 2 (O)

```
SELECT DISTINCT Magaja.* FROM Magaja, Magamine WHERE
Magaja.magaja_id = Magamine.magaja_id;
```

Oracle

Täitmise aeg: -

Täitmisplaan: -

PostgreSQL

Täitmise aeg: 2577 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Join*). Seejärel sorteeritakse tulemus (*external merge*) ning eemaldatakse korduvad read (*Unique*).

Lause 2 kommentaar: Oracle ei ole nõus antud lauset täitma, sest tulemuses sisalduv tabeli *Magaja* veerg `kommentaar` on andmetüüpi LONG ja sellele ei ole võimalik rakendada korduste eemaldamist (`DISTINCT`).

ORA-00997: illegal use of LONG datatype

Eksperiment näitas, et sama probleem on ka Oracle uuema ning LONG asemel soovitatava andmetüübiga CLOB.

ORA-00932: inconsistent datatypes: expected - got CLOB

Lause 3 (O)

```
SELECT * FROM Magaja WHERE magaja_id IN (SELECT magaja_id
FROM Magamine);
```

Oracle

Täitmise aeg: 9670 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite poolühendamine (*HASH JOIN RIGHT SEMI*).

PostgreSQL

Täitmise aeg: 498 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*) ja tabelile *Magaja* loodud indeksit `pk_magaja_magaja` (*Index Scan*), et läbi viia tabelite poolühendamine (*Merge Semi Join*).

Kommentaar: Oracle valib esimese ja kolmanda lause täitmiseks sama täitmisplaani, kuid PostgreSQL tulemus on kiirem.

2.12 Poolvahe leidmine

Ülesanne: Leia kõik magajad, kellega pole seotud mitte ühtegi magamist.

Lause 1 (E)

```
SELECT M.magaja_id, M.eesnimi, M.perenimi, M.sugu, M.synni_aeg, M.aadress, M.telefon FROM Magaja M LEFT JOIN Magamine Ma ON M.magaja_id = Ma.magaja_id
GROUP BY M.magaja_id, M.eesnimi, M.perenimi, M.sugu, M.synni_aeg, M.aadress, M.telefon
HAVING Count(Ma.magaja_id)=0;
```

Oracle

Täitmise aeg: 1160 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite välisühendamine (*HASH JOIN RIGHT OUTER*). Seejärel grupeeritakse tulemus (*HASH GROUP BY*) ning leitakse read, kus on täidetud tingimus `Count(Ma.magaja_id)=0`.

PostgreSQL

Täitmise aeg: 1938 ms

Täitmisplaan: Süsteem viib läbi tabeli *Magaja* täieliku läbiskaneerimise, loob loetud väärtuste põhjal räsitabeli, viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja ühendab tabelid omavahel (*Hash Right Join*). Seejärel sorteeritakse ja grupeeritakse tulemus (*external merge*) ning leitakse read, kus on täidetud tingimus `Count(Ma.magaja_id)=0`.

Lause 2 (O)

```
SELECT Magaja.* FROM Magaja LEFT JOIN Magamine ON Magaja.magaja_id = Magamine.magaja_id WHERE Magamine.magaja_id IS NULL;
```

Oracle

Täitmise aeg: 6540 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite poolvahe (*HASH JOIN RIGHT ANTI*).

PostgreSQL

Täitmise aeg: 463 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*Index Only Scan*) ja tabelile *Magaja* loodud indeksit `pk_magaja_magaja` (*Index Scan*), et läbi viia tabelite poolvahe leidmine (*Merge Anti Join*).

Lause 3 (O)

```
SELECT * FROM Magaja WHERE magaja_id NOT IN (SELECT magaja_id FROM Magamine WHERE magaja_id IS NOT NULL);
```

Oracle

Täitmise aeg: 6320 ms

Täitmisplaan: Süsteem kasutab tabelile *Magamine* loodud indeksit `ak_magamine_magaja_algus` (*INDEX FAST FULL SCAN*) ja tabeli *Magaja* täielikku läbiskaneerimist, et läbi viia tabelite poolvahe (*HASH JOIN RIGHT ANTI*).

PostgreSQL

Täitmise aeg: rohkem kui 10 min

Täitmisplaan: Süsteem viib läbi tabeli *Magamine* täieliku läbiskaneerimise ja leiab kõik read, kus on täidetud tingimus `magaja_id IS NOT NULL`. Vahetulemus materialiseeritakse. Seejärel kasutatakse tabeli *Magaja* täielikku läbiskaneerimist, et leida kõik read, kus `magaja_id` ei sisaldu vahetulemuses.

Kommentaar: Oracle valib teise ja kolmanda lause täitmiseks sama täitmisplaani. Oracle täidab ebaotstarbeka lause (Lause 1) kiiremini kui teise täitmisplaaniga laused (Lause 2 ja Lause 3). Võib järeldada, et Oracle välisühendamise meetod *HASH JOIN RIGHT OUTER* on kiirem kui poolvahe leidmise meetod *HASH JOIN RIGHT ANTI*. Kolmas lause võtab PostgreSQL puhul tulemuse väljastamiseks väga kaua aega.

2.13 Tulemuste kokkuvõte

Tabelis 6 esitatakse uuringu koondtulemus. Iga vaadatud ülesande kohta tuuakse nii Oracle kui PostgreSQL jaoks välja kõige aeglasem ja kõige kiirem päringu täitmine. Kõik ajad on millisekundites (v.a juhtumid, kus lause täitmine võtab rohkem kui 10 minutit). Tulemustest on ilmekalt näha, kuidas ülesande erinevate lahenduste (erineva süntaksiga SQL lausete) korral leiab andmebaasisüsteem vastuse erineva kiirusega. Tulemustest kooruvad välja ka Oracle ja PostgreSQL töökiiruse erinevused. Kõiki ülesandeid lahendati samas serveris loodud andmebaasides ning andmebaasides olid ühesugused andmed.

Tabel 6: Koondtulemus

Andmebaasisüsteem	Oracle		PostgreSQL	
Ülesanne	Aeglaseim	Kiireim	Aeglaseim	Kiireim
Skalaarse funktsiooni väljakutse	60	10	126	0
Korduste eemaldamine	1180	850	520	90
Läbimõtlemata tingimused (tingimuse veerul pole indeksit)	150	120	166	38
Läbimõtlemata tingimused (tingimuse veerul on indeks)	70	70	2	2
Liigsetest tabelitest andmete küsimine	570	550	589	284
Liigsetest tabelitest andmete küsimine (2)	9470	9450	827	525
Probleemid grupeerimisega	+10 min	1010	1186	898
LEFT JOIN vs. INNER JOIN	620	360	1155	1020
Korduste eemaldamine (2)	990	950	1454	1163
Korduste eemaldamine (3)	890	860	1298	922
Tingimused, mis välistavad indeksi kasutamise	170	130	18	6
Poolühendamine	9890	9670	2577	498
Poolvahe leidmine	6540	1160	+10 min	463

Uuringutest tuli välja, et leidus ebaotstarbekaks peetud lause (vt jaotis 2.12), mis andis Oracle andmebaasisüsteemis vastuse kiiremini kui otstarbekaks peetud lause. Samuti selgus, et nii PostgreSQL kui Oracle suudavad indeksit kasutada ka olukordades, mis

võiksid sundida andmebaasisüsteemi optimeerimismoodulit valima indeksi kasutamise asemel tabeli täieliku läbiskaneerimise.

Vaadatud näidete põhjal saab Oracle ebaotstarbekalt kirjutatud lausete täitmisplaanide lihtsustamisega paremini hakkama kui PostgreSQL ning kasutab rohkem indekseid, kuid vaatamata sellele leiab PostgreSQL sageli päringu tulemused kiiremini.

3. Kokkuvõte

Töö eesmärgiks oli uurida, kui palju suudavad Oracle ja PostgreSQL andmebaasisüsteemid ebaotstarbekaid SQL päringuid (SELECT lauseid) optimeerida ja võrrelda selles kitsas aspektis neid andmebaasisüsteeme omavahel.

Töö viidi läbi Oracle Database 12c Enterprise Edition Release 1 ja PostgreSQL 9.3 andmebaasisüsteemide põhjal, mis asuvad samas serverarvutis. Töö tulemusena loodi andmebaas, genereeriti testandmed, käivitati SQL laused ning analüüsiti nende täitmise aega ja valitud täitmisplaane.

Töö tulemustest võib järeldada, et Oracle oskab täitmisplaani paremini lihtsustada ja kasutab rohkem indekseid kui PostgreSQL. Kuid mõne ülesande puhul oleneb töökiirus siiski SQL lausest.

Mõlema andmebaasisüsteemi jaoks leidus vaadeldud näidetes üks SQL lause, mida nad ei suutnud mõistliku ajaga täita. Oracle puhul oli see lause, kus tabelite ühendamiseks kasutatakse otsekorrutist ja grupeerimist ning ühendamistingimus on HAVING klauslis. PostgreSQL puhul oli see lause, kus tuli läbi viia poolvahe, kasutades operaatorit NOT IN.

Töö käigus selgus tihti, et Oracle suudab küll paremini täitmisplaani lihtsustada, kuid PostgreSQL täidab lause siiski kiiremini. Töö edasiarendusena võiks uurida, miks see nii on.

4. Summary

The aim of this work was to find out how much can Oracle and PostgreSQL database management systems (DBMSs) optimize unpractical SQL queries (SELECT statements) and compare these systems in this narrow context.

The work was done based on Oracle Database 12c Enterprise Edition Release 1 and PostgreSQL 9.3 DBMSs that were in the same server. As a result of this work, a database was created, test data was generated, SQL statements were executed, and the time taken and execution plan were analyzed.

It turns out that Oracle uses indexes more often and can simplify execution plan better than PostgreSQL. However, there are tasks where execution speed depends on SQL statement used.

Both DBMSs had one SQL statement from the set of analyzed statements that they could not execute in reasonable time. For Oracle it was the statement where tables were joined using Cross join and grouping and the join condition was in HAVING clause. For PostgreSQL it was the statement where semidifference operation was conducted using the NOT IN operator.

During conducting the experiments for this work, it often turned out that Oracle can simplify execution plan better than PostgreSQL, but still PostgreSQL can execute the statement faster. As a continuation of this work, it could be investigated why this happens.

5. Kasutatud materjal

1. DB-Engines Ranking, May 2014. [WWW] http://db-engines.com/en/ranking (01.06.2014)

2. Eessaar, E., 2013, õppeaine Andmebaasid II õppematerjalid, Teema 7. SQL andmekäitluskeele lausete töötlemine ja optimeerimine

3. Eessaar, E., 2014, õppeaine Andmebaasid I õppematerjalid, Ebaotstarbekas SQL

4. Fernandez I., 2011, Day 4: The way you write your query matters, http://iggyfernandez.wordpress.com/2011/12/04/day-4-the-twelve-days-of-sql-there-way-you-write-your-query-matters/ (14.05.2014)

5. Oracle Database Online Documentation 12c. [WWW] http://docs.oracle.com/cd/E16655_01/index.htm (31.05.2014)

6. Pascal F., 1988, SQL redundancy and DBMS performance. Database Programming and
Design.

7. Pascal F., 2013, Language Redundancy and DBMS Performance: A SQL Story, http://www.dbdebunk.com/2013/02/language-redundancy-and-dbms.html (14.05.2014)

8. PostgreSQL 9.3 Documentation. [WWW] http://www.postgresql.org/docs/9.3/ (08.05.2014)

Lisa 1

```
package generator;

import java.io.PrintWriter;
import java.util.Random;

public class Generator {

	private static char[] symbols;
	private static char[] genderSymbols = {'M', 'N'};
	private static char[] booleanSymbols = {'0', '1'};

	private int rows;
	private String tableName;
	private Table table;
	private String fileName;

	private final Random random = new Random();

	static {
		StringBuilder builder = new StringBuilder();
		appendCharRange(builder, 'A', 'Z');
		appendCharRange(builder, 'a', 'z');
		appendCharRange(builder, '0', '9');
		builder.append("   ");
		symbols = builder.toString().toCharArray();
	}

	public Generator(int rows, String tableName) {
		this.rows = rows;
		this.tableName = tableName;
		table = getTableInstance(tableName);
		fileName = tableName+"_"+rows+".sql";
	}

	private static void appendCharRange(StringBuilder builder, char first,
char last) {
		for (char c = first; c <= last; c++) {
			builder.append(c);
		}
	}

	private Table getTableInstance(String tableName) {
		switch (tableName) {
		case "ase": return new Ase();
		case "magaja": return new Magaja();
		case "magamine": return new Magamine();
		case "mootmine": return new Mootmine();
		default:
			throw new IllegalArgumentException();
		}
	}

	private char getRandomChar(char[] symbols) {
		return symbols[random.nextInt(symbols.length)];
```

```
	}

	private int generateRandomInt(int min, int max) {
		return min + random.nextInt(max - min + 1);
	}

	private float generateRandomFloat(float min, float max) {
		return min + ((max - min)*random.nextFloat());
	}

	private String generateRandomString(int length) {
		StringBuilder builder = new StringBuilder();
		for (int i = 0; i < length; i++) {
			builder.append(getRandomChar(symbols));
		}
		return builder.toString();
	}

	private String generateRandomString(int min, int max) {
		return generateRandomString(generateRandomInt(min, max));
	}

	private String generateRandomDate() {
		return generateRandomInt(1940, 2000)+"-"+generateRandomInt(1,
12)+"-"+generateRandomInt(1, 28);
	}

	private String generateRandomDateTime() {
		return generateRandomDate()+" "+generateRandomInt(0,
23)+":"+generateRandomInt(0, 59);
	}

	private String getFirstLine() {
		StringBuilder builder = new StringBuilder("INSERT INTO
"+tableName+" (");
		boolean first = true;
		for (String column : table.getColumns()) {
			if (!first) {
				builder.append(",");
			}
			builder.append(column);
			first = false;
		}
		builder.append(") VALUES ");
		return builder.toString();
	}

	private String getValueLine() {
		StringBuilder builder = new StringBuilder("(");
		boolean first = true;
		for (String value : table.getValues()) {
			if (!first) {
				builder.append(",");
			}
			builder.append(value);
			first = false;
		}
		builder.append(")");
		return builder.toString();
```

```
	}

	private void generate() {
		try {
			PrintWriter writer = new PrintWriter(fileName);
			try {
				String firstLine = getFirstLine();
				for (int i = 0; i < rows; i++) {
					writer.print(firstLine);
					writer.print(getValueLine());
					writer.println(";");
				}
			} catch (Exception e) {
				throw e;
			} finally {
				writer.close();
			}
			System.out.println("Tabeli '"+tableName+"' andmed kirjutati
faili '"+fileName+"' ("+rows+" rida)");
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public static void main(String[] args) {
		if (args.length < 2) {
			System.out.println("Parameetrid: [ridade arv] [tabeli
nimi]");
			return;
		}
		int rows = Integer.parseInt(args[0]);
		String tableName = args[1] ;
		Generator generator = new Generator(rows, tableName);
		generator.generate();
	}

	private abstract class Table {
		protected abstract String[] getColumns();
		protected abstract String[] getValues();
	}

	protected class Ase extends Table {

		private final String[] columns = {
			"nimi", "pikkus", "laius", "korgus", "ostmise_kuupaev",
"eksisteerib", "vaba", "kommentaar"
		};

		@Override
		protected String[] getColumns() {
			return columns;
		}

		@Override
		protected String[] getValues() {
			String[] result = new String[columns.length];
			int i = 0;
			result[i++] = "'"+generateRandomString(5, 45)+"'";
			result[i++] = ""+generateRandomInt(1, 299);
```

```
                result[i++] = ""+generateRandomInt(1, 299);
                result[i++] = ""+generateRandomInt(1, 109);
                result[i++] = "'"+generateRandomDate()+"'";
                result[i++] = "'"+getRandomChar(booleanSymbols)+"'";
                result[i++] = "'"+getRandomChar(booleanSymbols)+"'";
                result[i++] = "'"+generateRandomString(5, 200)+"'";
                return result;
            }

        }

        protected class Magaja extends Table {

            private final String[] columns = {
                "eesnimi", "perenimi", "sugu", "synni_aeg", "aadress",
"telefon", "kommentaar"
            };

            @Override
            protected String[] getColumns() {
                return columns;
            }

            @Override
            protected String[] getValues() {
                String[] result = new String[columns.length];
                int i = 0;
                result[i++] = "'"+generateRandomString(5, 45)+"'";
                result[i++] = "'"+generateRandomString(5, 45)+"'";
                result[i++] = "'"+getRandomChar(genderSymbols)+"'";
                result[i++] = "'"+generateRandomDate()+"'";
                result[i++] = "'"+generateRandomString(10, 100)+"'";
                result[i++] = "'"+generateRandomString(8, 25)+"'";
                result[i++] = "'"+generateRandomString(5, 200)+"'";
                return result;
            }

        }

        protected class Magamine extends Table {

            private final String[] columns = {
                "ase_id", "magaja_id", "algus", "kestus", "kommentaar"
            };

            @Override
            protected String[] getColumns() {
                return columns;
            }

            @Override
            protected String[] getValues() {
                String[] result = new String[columns.length];
                int i = 0;
                result[i++] = ""+generateRandomInt(25, 100000);
                result[i++] = ""+generateRandomInt(25, 100000);
                result[i++] = "'"+generateRandomDateTime()+"'";
                result[i++] = ""+generateRandomInt(1, 999);
                result[i++] = "'"+generateRandomString(5, 200)+"'";
```

```
			return result;
		}

	}

	protected class Mootmine extends Table {

		private final String[] columns = {
			"magaja_id", "mootmise_aeg", "pikkus", "kaal"
		};

		@Override
		protected String[] getColumns() {
			return columns;
		}

		@Override
		protected String[] getValues() {
			String[] result = new String[columns.length];
			int i = 0;
			result[i++] = ""+generateRandomInt(25, 100000);
			result[i++] = "'"+generateRandomDateTime()+"'";
			result[i++] = ""+generateRandomInt(1, 249);
			result[i++] = ""+generateRandomFloat(1, 599);
			return result;
		}

	}

}
```