TARTU ÜLIKOOL

Arvutiteaduse instituut
Informaatika õppekava

Anastassia Ivanova

SQL päringute koostamine, analüüsimine ja optimeerimine *SQL Anywhere* 17.0 versiooni näitel

Bakalaureusetöö (9 EAP)

Juhendaja: MSc Ljubov Jaanuska

TARTU 2017

SQL päringute koostamine, analüüsimine ja optimeerimine *SQL Anywhere* 17.0 versiooni näitel

Lühikokkuvõte

Antud bakalaureusetöö eesmärk on koostada referatiivne eestikeelne lisamaterjal ainele „Andmebaasid“ (ilma lisaülesannete kogumikuta). Töös vaadeldakse kolme põhiaspekti: päringute koostamine, indekseerimine ja päringute analüüsimine päringu planeerija *Plan Viewer*-i abil. Igas osas on toodud praktilised näited, mis on tehtud kasutades andmebaasi „Film“ ning andmebaasi juhtimissüsteemi *SQL Anywhere 17.0*. Koostatud materjal on mõeldud eelkõige esimese kursuse informaatika õppekava üliõpilastele. Motivatsioon bakalaureusetöö kirjutamiseks on *SQL Anywhere* eestikeelse materjali puudumine, mis muudab üliõpilastel õppimist keerulisemaks. Antud töös ei ole käsitletud andmebaasi kasutamist kahel serveril. See võib olla antud töö edasiarendus.

Võtmesõnad:

SQL Anywhere 17.0 Developer Edition, päringud, indekseerimine, *Plan Viewer*, optimeerimine, restruktureerimine

CERCS: P170, S281

Structuring, analyzing and optimization of SQL queries using SQL Anywhere 17.0 version

Abstract

The goal of this thesis is to develop studying materials for undergraduate students about construction, analyzing and optimization of SQL statements in SQL Anywhere 17.0 Developer Edition. Motivation for writing this thesis was a shortage of the materials in Estonian about SQL Anywhere. This thesis addresses three main aspects: query modelling, indexes and query optimization. Also there is an introduction into the Plan Viewer tool in the thesis which helps analyze SQL query and reconstruct it in order to increase the database management system perfomance. Each section of the thesis has practice examples which were made using database „Film“. A student, who has read this material, can outline differences between various SQL operators, understands principles of indexes, lists pros and cons of indexes, knows how to analyze and optimize queries using the Plan Viewer tool. The outcome of the thesis is the studying materials with practical samples composed for database related courses. The result of this work can be used as additional literature for course „Databases“ (practices 3, 6 and 13). A further development of the thesis would be to conduct a research on how to distribute large databases between database servers and how to connect these servers efficiently.

Keywords:

SQL Anywhere 17.0 Developer Edition, query, index, Plan Viewer, optimization, restructuring

CERCS: P170, S281

Sisukord

1. Sissejuhatus

1.1 Probleemi püstitus

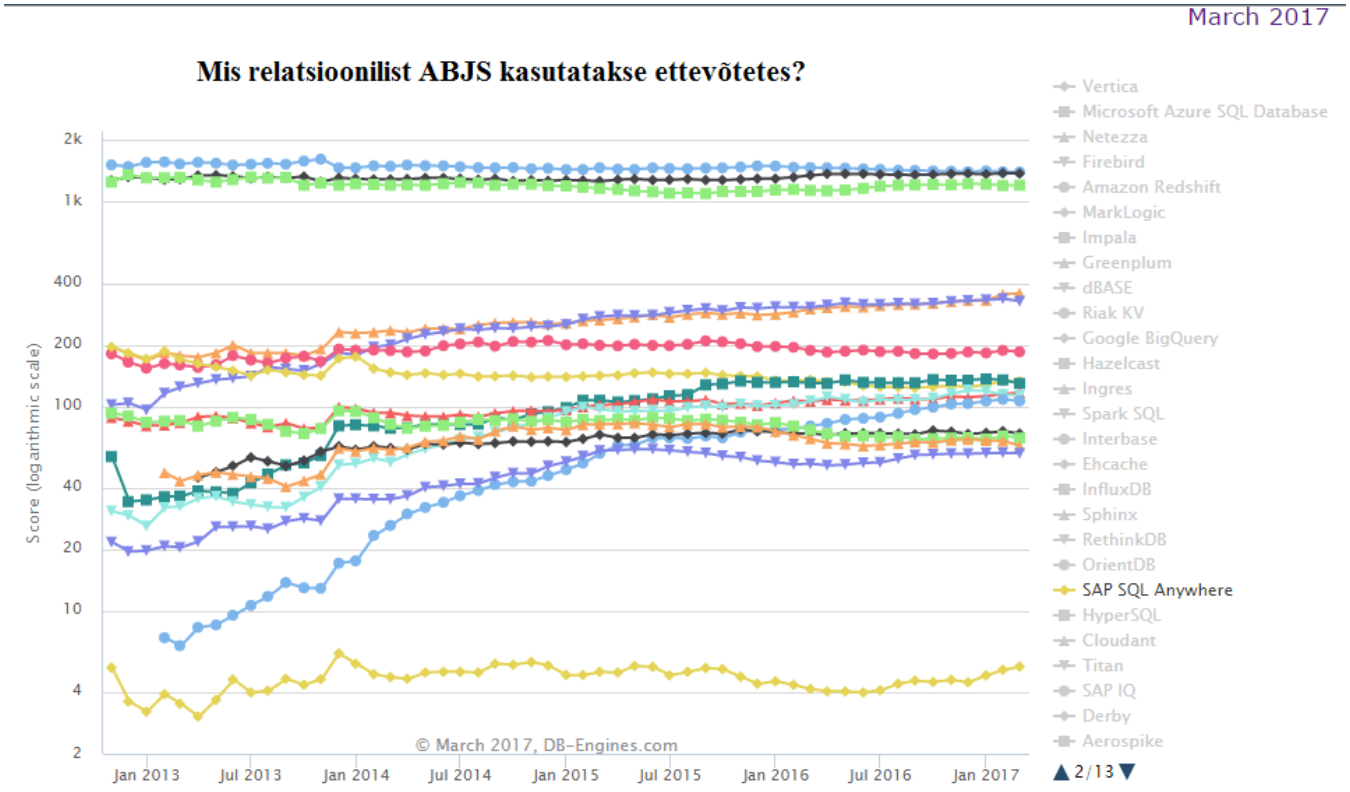
Eesti on üks edukatest e-riikidest, kus aktiivselt arendatakse ja kasutatakse e-teenuseid ning püüakse digitaliseerida andmeid. Andmete haldamiseks on loodud andmebaasid ning nende manipuleerimiseks kasutatakse erinevaid andmebaaside juhtimissüsteeme. Tänapäeval on andmebaasid aktiivselt kasutuses ning vajadus nende järgi kasvab igapäevaselt. Näiteks, andmebaasis hoitakse rahvastikuregistri andmeid, mille maht peab olema piisavalt suur, et sinna mahuksid andmed iga elaniku kohta. Mida suurem on rahvaarv, seda suurem peab olema andmebaas. Mida suuremad on andmete mahud, seda aeglasem ja keerulisem on neid lisada, kustutada, uuendada, sorteerida ja otsida. Sellel põhjusel tuleb lähtuda optimeerimise põhimõttest andmebaasi modelleerimisel, arendamisel ning kasutamisel. Indeksid, SQL lausete optimeerimise plaan, päringu puu ning päringute ümberehitamine on ainult vähesed märksõnad, mis aitavad töös andmebaasiga suurendada kiirust ning vähendada ootamisaega. Optimeerimise võimalusi on palju, kuid andmebaasi maksimaalse töökiiruse saavutamiseks on tarvis teada, mis on optimeerimine (vt. punkt 1.2.2) ja mis on selle põhimõte. Samas, et saavutada parimat tulemust, tuleb teada, kuidas efektiivselt salvestada, lisada ja kustutada andmeid. Selleks, et kasutada andmebaase maksimaalse kasuga, tuleb teada, millised võimalused on olemas andmebaasi juhtimissüsteemis ja millal on vaja neid kasutada.

Aines „MTAT.03.105 Andmebaasid“ ja „MTAT.03.264 Andmebaasid“ antakse ülevaade üldlevinud andmebaaside päringukeelest SQL ning andmebaaside juhtimissüsteemist *SQL Anywhere 17.0*. Kursuste ja loengute põhieesmärgiks on anda tudengitele vajalikud teadmised relatsioonilise andmebaaside projekteerimisest. Praktikumides näidatakse SQL süntaksit, koostatakse kolm andmebaasi ning arendatakse struktuur-loogilist mõtlemist. Eelnevat töökogemust andmebaasidesse sissejuhatavas aines ei eeldata. Aine on kohustuslik informaatika õppekaval õppivatele bakalaureuseüliõpilastele ning see on lihtne iseseisvalt edasijõudnud üliõpilaste jaoks. Antud materjal aitab üliõpilastel täiendada päringute koostamise ja optimeerimise oskust.

1.2 Kirjanduse ülevaade

1.2.1 Kirjanduse ülevaade

Alustuseks tuleb eristada kahte mõistet: andmebaasi juhtimissüsteem, mille abil manipuleeritakse andmebaasiga ja andmebaas, mis on täidetud andmetega. Andmebaas on andmete kogum, mis lisaks andmetele sisaldab andmete struktuuri kirjeldust. Tarkvara, mille abil andmebaase luuakse ja hallatakse nimetatakse andmebaaside juhtimissüsteemiks e. **ABJS** *(ingl.k. database Management System e. DBMS)* [1]. Läbi aegade on loodud mitmeid erinevaid andmebaaside juhtimissüsteeme. Enim kasutatud süsteemid on näidatud pildil 1. Uuring oli tehtud aastal 2017.



Pilt 1. Andmebaasijuhtsüsteemide populaarsus üle maailma 2017. a. [2].

Ainetes „MTAT.03.105 Andmebaasid“ ja „MTAT.03.264 Andmebaasid“ kasutatakse *SQL Anywhere 17 Developer Edition*. *SAP* on üks suurim tarkvaraarenduse firma, millel on üle 100 tütarettevõtte, üle 53000 töötaja ja üle 109 000 kliendi ning 2016 aasta käive oli 22,062 miljardit eurot [4]. *SQL Anywhere*'i algseks nimeks võib pidada *Watcom SQL*'i, mis loodi 1992 *Watcom*'i poolt [3]. Süsteemil on

pikk ajalugu ja arengu käigus on kasutatud mitmeid nimesid: *SQL Anywhere Studio, Adaptive Server Anywhere*.

2006. aastal tuli välja kümnes versioon ja sellega muudeti taas nime, milleks sai *SQL Anywhere* [3]. Aastal 2010 avalikustati versioon 12.0.0 ja viis aastat hiljem avalikustati versioon 17.0.0, mis on hetkel viimane versioon.

Tänapäevaks on *SQL Anywhere*'ist saanud terviklahenduse, mis pakub andmehalduse, sünkroniseerimise ja andmete vahetamise tehnoloogiaid [3]. Lisaks *SQL Anywhere*'ile kuulub *SQL Anywhere* paketti ka *MobLink, UltraLite* ja *Relay Server*. *SQL Anywhere* omab veebirakenduste loomiseks paljudele programmeerimiskeeltele tuge nagu näiteks *PHP, Python, Ruby on Rails, AJAX, Java, ASP.NET, Perl, Flex* ja *Silverlight* [3]. *SQL Anywhere* toetab ka *XML*'i, veebiteenuseid ja tervikteksti (*full text*) otsingut [3].

SQL Anywhere'ist on väljas seitse erinevat versiooni [3]:

1. *Developer;*
2. *Educational;*
3. *Web;*
4. *OEM;*
5. *Workgroup;*
6. *Standard;*
7. *Advanced.*

Ainetes „MTAT.03.105 Andmebaasid" ja „MTAT.03.264 Andmebaasid" valiti *SQL Anywhere Developer 17.0* töö vahendina mitmetel põhjustel:

1. see on tasuta rakendus õppeotstarveks;
2. see on üks populaarsematest andmebaasi juhtimissüsteemidest tänapäeval;
3. see tarkvara nõuab tunduvalt vähem mälu, kui, näiteks Oracle tarkvara;
4. see on üks esimestest, kes pakkus klient-server relatsioonilist andmebaasi juhtimissüsteemi lahendust.

SQL Anywhere avaldas ka oma toodete inglisekeelse dokumentatsiooni [6], millesse on lisatud märkused optimeerimise kohta. See kirjandus annab üldise ülevaate andmebaasist ning enim kasutatavatest meetoditest. Samuti sisaldab informatsiooni alatest tutvustusest ja lõpetades optimeerimisega.

1.2.2 Optimeerimine

Andmebaasi jõudluse suurendamine on keeruline valdkond, mida saab uurida, lähtudes erinevatest vaatepunktidest. Selleks, et vähendada päringu täitmise aega, kasutatakse optimeerimist, mille esmane eesmärk on kas vähendada mälu kasutust või vähendada lahendamise aega.

Tavaliselt ei optimeerita programmi lähteteksti tasandil. Mida kõrgem on keele tase (skaalal: masinkood, assembler, kõrgtaseme keel, väga kõrge taseme keel), seda paremini töötab **automaatne optimeerimine** [5]. Optimeerimine on protsess, mille käigus elimineeritakse toimuvaid kõrvalprotsesse andmebaasi juhtimissüsteemi töö käigus. SQL programmi optimeerimise põhimõtted:

- teksti kujul päring ei sobi optimeerimiseks;
- SQL-päring teisendatakse sisekujule (sisemisele **puu** kujule);
- puu esitus põhineb relatsioonalgebral;
- üldidee: teisendada päringu puu esialgsega ekvivalentseks puuks, rakendades optimeerivaid teisendusi;
- kasutades lisainfot valitakse täitmise algoritm.

Peale nimetatud põhimõtteid, enamus andmebaaside juhtimissüsteeme kasutavad ühist optimeerimise teooriat. „*Top 10 performance tuning tips for relational databases*“ [7] juhendis on olemas üldised optimeerimise nõuanded, mis kehtivad enamuse andmebaasi juhtimissüsteemi korral ning mida saavad kasutada kõik andmebaaside administraatorid ja arendajad. Selliste üldiste soovituste juurde kuuluvad:

- andmebaasi statistika kasutamine, mille abil saab optimeerija leida kõige säästlikuma viisi päringute töötlemiseks;
- indeksite kasutamine;
- `SELECT` päringute oskuslik kirjutamine;
- välisvõtmete kasutamine.

SQL Anywhere andmebaasi juhtimissüsteemis on loodud päringu planeerija, mille abil saab:

- vähendada SQL päringute töötlemise aega, kasutades abstraktseid plaane ja indekseid;
- suurendada efektiivsust;
- parandada rakenduse jõudlust, mis positiivselt mõjutab **rakenduse elutsüklit** [8].

SQL Anywhere päringu planeerija kohta on kirjutatud pikemalt peatükis 5.

1.3 Eesmärgi püstitus

Antud töö eesmärk on koostada eestikeelne lisamaterjal ainetele „MTAT.03.105 Andmebaasid“ ja „MTAT.03.264 Andmebaasid“ *SQL Anywhere* kohta. Töös vaadeldakse kolme aspekti:

1. SQL päringute koostamine;
2. indekseeritud tabelite koostamine ja nende mõju andmebaasi juhtimissüsteemi jõudlusele;
3. päringu planeerija *Plan Viewer* kasutamine.

Esimeses peatükis kirjeldatakse metoodikaid, mis on kasutatud antud õppematerjali koostamisel.

Teise peatüki eesmärk on koostada juhend, kuidas kirjutada päringuid nii, et töötlemiseks kuluks võimalikult vähem aega. Antakse nõuandeid, kuidas kirjutada optimaalsemat päringut ning tuuakse näidisolukordi, kus erinevate meetmete kasutamine parandab andmebaasi jõudlust. Lisaks tuuakse optimeeritud päringute näiteid ning nende võrdlus optimeerimata päringutega ajakulu järgi.

Kolmandas peatükis on tehtud ülevaade *SQL Anywhere* andmebaasi jõudluse suurendamisest indekseeritud tabelite abil. Selles osas antakse ülevaade indeksitest ja nende tüüpidest ning seletatakse, kuidas ja millistel juhtudel neid kasutada. Samuti loetletakse indeksite eeliseid ja puuduseid. Praktilises osas analüüsitakse päringute töötlemise aega indekseeritud ja indekseerimata tabelite kasutamisel.

Viimases peatükis on kirjeldatud *SQL Anywhere* päringu planeerija, selle omadused ning seletatud kuidas analüüsida ja parandada koostatud päringut päringu planeerija abil. Lisaks on kirjeldatud, kuidas käib päringute läbimine, optimeerimise plaani koostamine ja selle täitmine. Praktiline osa seisneb selles, et analüüsitakse päringut ning tuuakse välja selle kitsaskohad.

Lõputöö tulemuseks on *SQL Anywhere 17.0* õppematerjal. Tulevikus on võimalik antud töö täiendamiseks uurida kuidas kastutada *SQL Anywhere*'i kahel või rohkematel serveritel. Samuti saab uurida, kuidas ühendada erinevate andmebaaside juhtimissüsteemide tööd (näiteks, *Oracle* ja *SQL Anywhere)*.

2. Kasutatud skoop ja metoodika

Antud materjal annab tudengitele rohkem võimalusi SQL päringute optimeerimise õppimiseks. Käesolevas töös on koostatud referatiivne eestikeelne lisamaterjal aine „Andmebaasid“ 3., 6. ja 13. praktikumitele. Antud töö ei eeldanud lisaülesannete kogumiku koostamist. Töö on jagatud kolmeks põhiteemadeks:

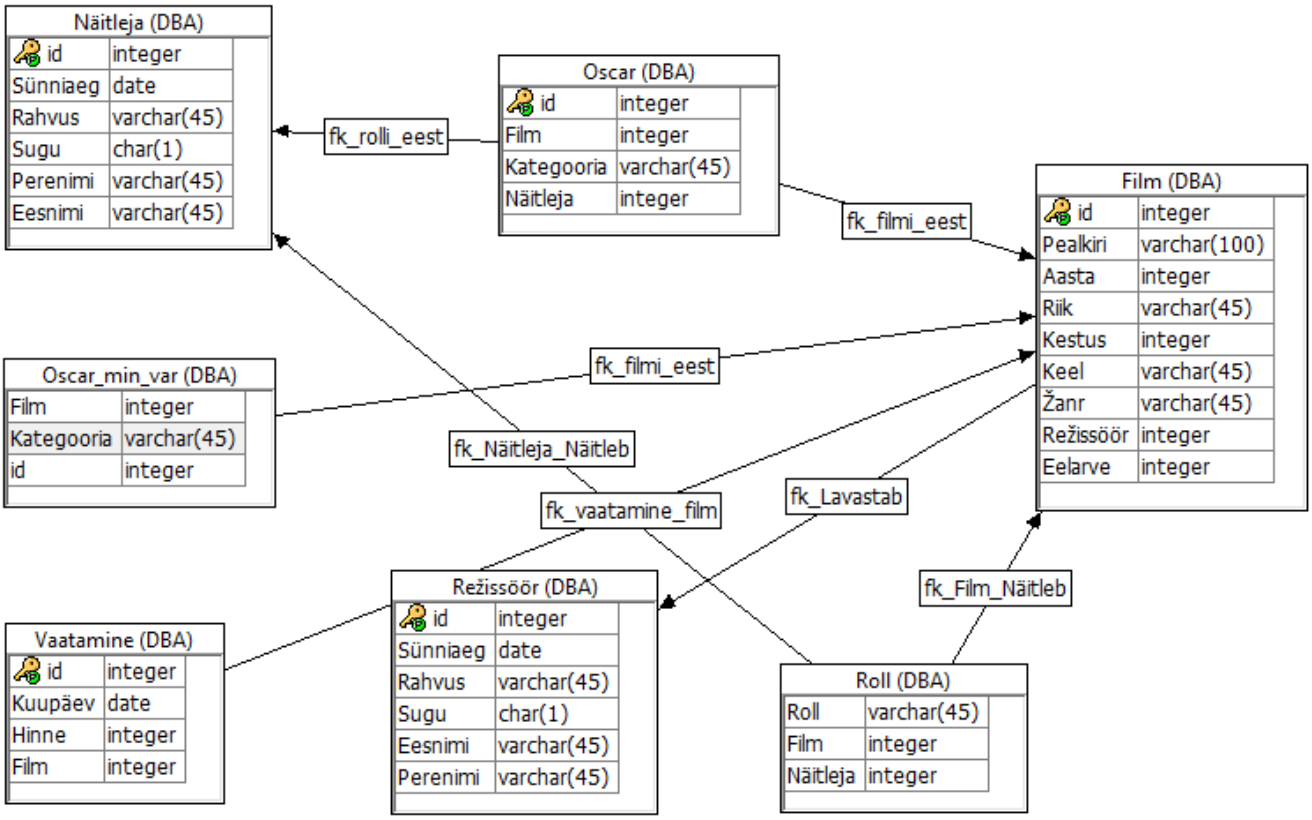
- päringute koostamise üldised reeglid;
- indekseerimine;
- päringu analüüsimine päringu planeerija abil.

Kokku need kolm osa annavad tudengitele ülevaate *SQL Anywhere* tööprintsiibist, õpetab kirjutama efektiivseid SQL päringuid ning näitab, kuidas analüüsida koostatud päringut ja välja tuua selle kitsaskohad. Töö koostamisel olid kasutusele võetud järgmised teooriad:

- andmebaaside teooria;
- normaalkujude teooria;
- relatsioonalgebra;
- relatsioonarvutus;
- hulgateooria.

Materjali teoreetilise osa valmimine käis paralleelselt praktiliste katsetega, mille käigus on uuritud erinevad optimeerimise meetodeid. Iga SQL lause juures on töötlemise aeg.

Töö praktilise osa jaoks on loodud andmebaas nimega „Film“ ning see andmebaas koondab endas informatsiooni filmide kohta, mida kasutaja on näinud. Korralik filmisõber vaatab väga palju filme ning seetõttu andmebaas on hea abimees meenutamaks, millist filmi filmisõber vaatas eelmise aasta sõbrapäevakohtingul, millised Kubricku filmid on juba vaadatud, mitu korda on filmisõber vaadanud "Terminaatorit" jne. Ühtlasi saab kasutaja panna vaadatud filmidele hindeid. Allpool toodud SQL Anywhere-i E-R mudel:



Pilt 2. Andmebaasi „Film“ graafiline mudel.

Mudelis on tugevad olemid Film, Vaatamine, Isik, Näitleja ja Oscar. Andmebaasi relatsiooni kirjeldused ja seosed on toodud Lisas 4.

Andmebaas sisaldab tabeleid „Režissöör“, „Film“, „Vaatamine“, „Oscar“, “Oscar_min_var”, „Näitleja“ ja „Roll“, mille loomise päringud on Lisas 2. Igas tabelis on rohkem, kui 10 000 rida, kus 250 andmete rida on täidetud käsitsi koostatud andmetega ning ülejäänud read on täidetud juhuslikult genereeritud andmetega. Lisas 3 on toodud tabelite täitmise SQL laused ja juurde loodud funktsioonid, mis olid kasutatud selleks, et täita tabeleid andmetega.

3. Päringute koostamine

Antud peatükis vaadetakse, kuidas koostada SQL päringuid nii, et nende töötlemine võtaks võimalikult vähem aega. Uuritakse, kuidas teha olemasolevaid päringuid kompaktsemaks ning kuidas kasutada sisseehitatud funktsioone. Samuti antakse ülevaade, kuidas muuta andmeid tabelites nii, et nende kättesaamine oleks kiirem.

SQL ei ole programmeerimise, vaid andmete manipuleerimise keel. Seega SQL keeles pole käske, vaid on olemas laused (*statements*) nagu `INSERT`, `DELETE`, `CREATE` ja `UPDATE` ning päringud (*queries*) `SELECT`. Allpool on toodud näited kõige lihtsamatest tehnikatest, mis aitavad teha päringut täpsemaks ja kiiremaks. Päringu sõnastamisel on oluline oskuslik otsingu tingimuste kombineerimine, et piiritleda otsingut võimalikult selgelt ja täpselt [9].

3.1 Boole´i loogikaoperaatorid (`AND`, `OR` ja `NOT`)

Mida täpsemini on päringus kirja pandud otsingu parameetrid, seda rangem on andmete filtreerimine. Mida rangem on andmete filtreerimine, seda täpsem on tagastatud tulemus ning seda vähem aega võtab päringu töötlemine.

- **`AND:`** tulemuse saamiseks peavad kehtima kõik tingimused mis on ühendatud antud operaatoriga [9]. Näide on toodud päringus 1.

  Päring 1:

```
SELECT * FROM Vaatamine WHERE Film < 2 AND Hinne > 5;
```

  - Execution time: 0.094 seconds

  Antud päringu puhul tulemuse hulka kuuluvad filmid, mille id on väiksem kui kaks või mille hinne on suurem kui viis. Päringu 1 tulemus on toodud allpool oleval pildil 3.

	id	Kuupäev	Hinne	Film
1	1	2015-01-13	7	1
2	3 603	1996-02-10	580	1
3	3 808	1997-03-15	185	1
4	4 227	1996-06-25	814	1
5	6 220	1974-11-18	16	1
6	7 885	1977-12-11	675	1
7	8 736	1993-04-23	846	1

Pilt 3. Päringu 1 tulemus.

- **OR** : tulemuse saamiseks peab vähemalt üks tingimustest kehtima [9]. Operaatoriga `OR` on otstarbekas ühendada:

  - Sünonüüme (vt. Päring 2)

    Päring 2:

```
SELECT * FROM Film WHERE Pealkiri = 'Terminator Salvation'
OR Pealkiri = 'The Terminator';
```

    - Execution time: 0.047 seconds

    Antud päringu puhul kuuluvad tulemuse hulka filmid, mille pealkiri on kas „Terminator Salvation“ või „The Terminator“. Päringu 2 tulemus on toodud allpool oleval pildil 4.

	id	Pealkiri	Aasta	Riik	Kestus	Keel	Žanr	Režissöör	Eelarve
1	10	The Terminator	1 984	Inglismaa	70	itaalia	kriminaal	74	145 112
2	12	Terminator Salvation	2 009	Inglismaa	116	inglise	sõjafilm	43	398 460

Pilt 4. Päringu 2 tulemus.

  - Sõnad, mille kirjutamine on sarnane (vt. Päring 3)

    Päring 3:

```
SELECT * FROM Režissöör WHERE Eesnimi = 'Rob' OR Eesnimi =
'Robert';
```

    - Execution time: 0.031 seconds

    Antud päringu puhul kuuluvad tulemuse hulka režissöörid, mille eesnimi on kas Rob või Robert. Päringu 3 tulemus on toodud allpool oleval pildil 5.

	id	Sünniaeg	Rahvus	Sugu	Eesnimi	Perenimi
1	34	1982-11-15	iirilane	F	Robert	Zemeckis
2	47	1985-02-23	kanadalane	M	Rob	Reiner
3	52	2003-06-19	mehhiklane	M	Robert	Altman
4	87	1950-10-24	austaallane	M	Robert	Wise
5	94	1982-05-18	prantslane	F	Robert	Rodriguez

Pilt 5. Päringu 3 tulemus.

- Sarnase tähendusega või seotud sõnu ja fraase (vt. Päring 4)

Päring 4:

```
SELECT * FROM Näitleja WHERE Rahvus = 'prantslane' OR Rahvus = 'inglane';
```

- Execution time: 0.063 seconds

Antud päringu puhul kuuluvad tulemuse hulka näitlejad, kes on rahvuse järgi kas prantslased või inglased. Päringu 4 tulemus on toodud allpool oleval pildil 6.

	id	Sünniaeg	Rahvus	Sugu	Perenimi	Eesnimi
1	1	1989-10-19	prantslane	F	Nicholson	Jack
2	3	1975-05-27	prantslane	F	De Niro	Robert
3	8	1986-02-10	inglane	M	Hopkins	Anthony
4	10	1985-11-26	inglane	F	Washington	Denzel
5	19	1991-01-29	prantslane	F	Seymour Hoffman	Philip
6	25	1991-12-09	prantslane	F	Poitier	Sidney
7	27	1982-01-09	prantslane	F	Hackman	Gene
8	35	1985-12-05	inglane	M	Lemmon	Jack
9	36	1984-03-01	inglane	M	C. Scott	George
10	38	1974-07-26	inglane	M	Robards	Jason
11	40	1980-12-30	inglane	M	Sellers	Peter
12	50	1987-05-11	prantslane	M	Plummer	Christopher
13	61	1984-04-03	inglane	M	Foxx	Jamie
14	75	1992-04-12	inglane	M	Neeson	Liam
15	77	1993-10-15	inglane	F	Kline	Kevin
16	78	1990-05-07	prantslane	F	Irons	Jeremy
17	80	1980-12-12	prantslane	M	Downey Jr.	Robert
18	81	1993-01-02	inglane	M	Clooney	George
19	83	1985-11-03	inglane	F	Gibson	Mel
20	97	1995-01-16	inglane	F	Smith	Will
21	100	1976-04-01	inglane	F	Costner	Kevin

Pilt 6. Päringu 4 tulemus.

- **NOT**: selle operaatori abil saab välistada kirjeid atribuutidele etteantud väärtustega [7] (vt. Päring 5)

  Päring 5:

```
SELECT * FROM Film WHERE Aasta NOT BETWEEN 2000 AND 2010;
```

  - Execution time: 0.032 seconds

  Päringu tulemuse hulgas on kõik filmid, va need, mis ilmusid aastatel 2000 kuni 2010. Kuna päringu 5 tulemus on väga mahukas (8439 rida), siis allpool oleval pildil 7 on toodud osa tulemusest.

	id	Pealkiri	Aasta	Riik	Kestus	Keel	Žanr	Režissöör	Eelarve
1	1	Fleshburn	1,984	Brasilia	101	saksa	fantaasia	176	339,413
2	2	First Blood	1,982	Kanada	105	hispaania	melodraama	68	262,172
3	3	Rambo: First Blood Part II	1,985	perefilm	95	itaalia	fantaasia	63	430,151
4	4	Rambo III	1,988	Argentina	64	inglise	sõjafilm	49	873,742
5	5	Raiders of the Lost Ark	1,981	Prantsusmaa	74	vene	draama	98	267,886
6	6	Beethoven	1,992	Austraalia	123	prantsuse	komöödia	204	904,019
7	7	Shinkansen daibakuha	1,975	Argentina	113	inglise	õudufilm	70	110,274
8	8	E.T. the Extra-Terrestrial	1,982	Prantsusmaa	124	itaalia	triller	72	613,871
9	9	Terminator 2: Judgment Day	1,991	Brasilia	107	hispaania	õudufilm	87	335,101
10	10	The Terminator	1,984	Inglismaa	70	itaalia	kriminaal	74	145,112
11	13	Commando	1,985	Argentina	162	itaalia	sõjafilm	21	721,876
12	14	Raw Deal	1,986	Kanada	148	vene	õudufilm	26	910,897
13	15	Red Heat	1,988	Austraalia	74	prantsuse	õudufilm	223	27,772
14	16	Predator	1,987	Kanada	175	itaalia	melodraama	162	517,211
15	17	Predator 2	1,990	Prantsusmaa	110	prantsuse	komöödia	236	314,350
16	18	Conan the Barbarian	1,982	Brasilia	125	vene	draama	60	134,915

Pilt 7. Päringu 5 osa tulemusest.

- **Sulgude kasutamine**

  Kui kasutada ühes päringus erinevaid operaatoreid, tuleb arvestada, et operaatoritel `NOT` ja `AND` on kõige kõrgem prioriteet ning operaatoril `OR` on kõige madalam [9]. Seega päringus vaadatakse esmaselt neid tingimusi, mis on ühendatud `NOT` ja `AND` operaatoritega ning viimasena vaadatakse `OR` operaatoriga ühendatud avaldisi. Kui on vaja operaatorite prioriteete muuta, tuleb kasutada sulge, sest sulgudes olevat avaldist kontrollitakse kõige ennem. On väga oluline panna sulud õigesse kohta, sest see mõjutab tulemust. Näiteks, kui sulud on pandud valesti, siis tulemuses võivad olla read, mis ei vasta päringu eesmärgile. Sulgude kasutamise näidis on toodud päringus 6.

Päring 6:

```
SELECT f.pealkiri, f.aasta, f.riik, f.žanr FROM  Film as f LEFT
JOIN Režissöör as r on f.režissöör = r.id WHERE (r.eesnimi =
'Anna' OR r.eesnimi = 'Joe') AND (r.perenimi = 'Foerster'  OR
r.perenimi = 'Wright');
```

- Execution time: 0.031 seconds

Kuna päringu 6 tulemus on mahukas (40 rida), siis pildil 8 on toodud osa tulemusest:

	pealkiri	aasta	riik	žanr
1	The Rescue	1 988	Brasilia	perefilm
2	Underworld: Rise of the Lycans	2 009	Venemaa	melodraama
3	The Ghost and the Darkness	1 996	Hispaania	romantika
4	Unstoppable	2 004	Venemaa	õudufilm
5	The Heartbreak Kid	2 007	Ameerika	romantika
6	Doom	2 005	Korea	seiklus
7	Crazy Eights	2 006	Hispaania	komöödia
8	The Bird with the Crystal Plumage	1 970	Prantsusmaa	romantika
9	Body Heat	1 981	Korea	õudufilm
10	The Free Will	2 006	Brasilia	triller
11	Repulsion	1 965	Hispaania	sõjafilm
12	Hanger	2 009	Itaalia	kriminaal
13	Murder on the Orient Express	1 974	Kanada	komöödia
14	Devil	2 010	Argentina	perefilm
15	Akai kami no onna	1 979	Venemaa	seiklus

Pilt 8. Päringu 6 tulemus.

Päringus 6 eelkõige kontrollitakse sulgudes olevaid tingimusi ehk kontrollitakse kõige ennem režissööri täisnime. Lubatud kombinatsioonid on järgmised:

- Anna Wright;
- Joe Foerster;
- Anna Foerster;
- Joe Wright.

Kui on vaja leida sama tüvega sõnu, on võimalik kasutada kärpimist, et lühendada päringu pikkust. Kärpimisel pannakse sõnadesse metamärke (vt punkt 3.3 Regulaaravaldised). Kasutamise näidis on toodud päringus 7.

Päring 7:

```
SELECT f.pealkiri, f.aasta, f.riik, f.žanr FROM  Film as f LEFT
JOIN Režissöör as r on f.režissöör = r.id WHERE (r.eesnimi like
'An%' OR r.eesnimi like 'Jo%') AND (r.perenimi like 'Fo%'  OR
r.perenimi like 'Wr%');
```

- Execution time: 0.1 seconds

Selline päring laiendab otsingu ruumi ning tulemus on suurem kui päringus 6, sest sobivad kõik režissöörid, kelle eesnimi ja perenimi algavad eeltoodud tähtetega. Kuna otsingu ruum on suurem, siis päringu töötlemine ja tulemuse tagastamine võtab rohkem aega. Siin võib teha järelduse, et kui on teada täpne väärtus, tasub kasutada võrdusmärki (=) või vastasel juhul tuleb kasutada aeglasemat varianti `LIKE`.

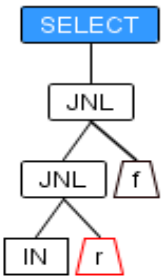
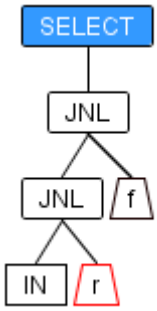
Kuna päringu 7 tulemus on mahukas (60 rida), siis pildil 9 on toodud osa tulemusest:

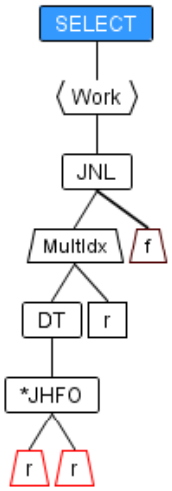
	pealkiri	aasta	riik	žanr
1	The Rescue	1 988	Brasilia	perefilm
2	Underworld: Rise of the Lycans	2 009	Venemaa	melodraama
3	The Ghost and the Darkness	1 996	Hispaania	romantika
4	Unstoppable	2 004	Venemaa	õudufilm
5	The Heartbreak Kid	2 007	Ameerika	romantika
6	Doom	2 005	Korea	seiklus
7	Crazy Eights	2 006	Hispaania	komöödia
8	The Bird with the Crystal Plumage	1 970	Prantsusmaa	romantika
9	Body Heat	1 981	Korea	õudufilm
10	The Free Will	2 006	Brasilia	triller
11	Repulsion	1 965	Hispaania	sõjafilm
12	Hanger	2 009	Itaalia	kriminaal
13	Murder on the Orient Express	1 974	Kanada	komöödia
14	Devil	2 010	Argentina	perefilm
15	Akai kami no onna	1 979	Venemaa	seiklus

Pilt 9. Päringu 7 tulemus.

Järgmisena võrreldakse Boole'i operaatoreid. Allpool on toodud tabel 1, kus on kirja pandud päringute täitmise aeg, puu struktuur ning tagastatud ridade arv. Selle tabeli abil on võimalik võrrelda Boole´i operaatorite tööd sekundites ning teha järeldust, milline operaatoritest on kõige aeglasem ja kõige kiirem. Lisaks, saab võrrelda erinevate operaatoritega päringute puustruktuure.

Tabel 1. Boole'i operaatoritega päringute võrdlus.

Rea number	Päring	Aeg (sec)	Päringu puu struktuur	Tagastatud ridade arv
1.	`SELECT f.pealkiri, f.aasta, f.riik, f.žanr FROM Film as f JOIN Režissöör as r on f.režissöör = r.id WHERE (r.eesnimi = 'Anna' OR r.eesnimi = 'Joe') AND (r.perenimi = 'Foerster' OR r.perenimi = 'Wright');`	0.272		40
2.	`SELECT f.pealkiri, f.aasta, f.riik, f.žanr FROM Film as f JOIN Režissöör as r on f.režissöör = r.id WHERE (r.eesnimi = 'Anna' AND r.perenimi = 'Wright') OR (r.eesnimi = 'Joe' AND r.perenimi = 'Wright') OR (r.eesnimi = 'Anna' AND r.perenimi = 'Foerster') OR (r.eesnimi = 'Joe' AND r.perenimi = 'Foerster');`	0.036		40

3.	SELECT f.pealkiri, f.aasta, f.riik, f.žanr FROM Film as f JOIN Režissöör as r on f.režissöör = r.id WHERE (r.eesnimi LIKE 'Anna%' OR r.eesnimi LIKE 'Joe%') AND (r.perenimi LIKE 'Foerster%' OR r.perenimi LIKE 'Wright%');	0.006		40

Tabeli analüüsides, tuli välja, et esimene päring osutus kõige ebaefektiivsemaks. Selle põhjuseks on, et päringus on kombineeritud operaatorid `AND` ja `OR` ning kasutati ka sulge operaatorite prioriteedi paika panemiseks. Koostatud päring peab tagastama täpse kirje, mis vastab kõikidele päringus olevatele tingimustele. Selleks, et vaadata kõik kirjeid läbi ning leida õiget, kulub palju aega.

Teises päringus kasutati operaatorid `AND` ja `OR`, kuid kõik vastuseks sobivad kombinatsioonid olid käsitsi välja kirjutatud. Kuna tingimusteks olid pandud kõik sobivad variandid, siis see lihtsustas otsingut, mille tõttu tulemust tagastati 0.272 / 0.036 = 7.5 korda kiiremini, kui esimese päringu puhul.

Kõige efektiivsem päring tabelis 1 on kolmas päring. Peale `AND` ja `OR` operaatorite kombineerimist oli kasutatud ka kärpimist. Kuigi kärpimine laiendab otsingu ruumi ning operaatorid `AND` ja `OR` kontrollivad ridade vastavust tingimustele, kolmas päring võttis kõige vähem aega. Esimene päring võtab 0.272 / 0.006 = 45 korda rohkem aega. Võrreldes päringute puid, tuled tähele panna, et kolmanda päringu puu on sügavam ja keerulisem, kui esimese ja teise päringute puud. Siit võib teha järelduse, et kärpimine mõjutab päringu puu struktuuri.

3.2 Fraasiotsing

Fraasiotsimiseks kasutatakse `CONTAINS` funktsiooni, mille süntaks on toodud allpool:

`CONTAINS ( column-name [,...], contains-query-string )`, kus

`column-name` - veeru nimi, kus tehakse otsingut;
`contains-query-string` – muster, mille järgi otsitakse sobivat kirjet.

Selle funktsiooni töö põhimõte on sarnane operaatoriga `LIKE` (vt 3.3 Regulaaravaldised). Üks erinevus `LIKE` and `CONTAINTS` vahel on selles, et `CONTAINS` otsing on tingimuspõhine (*term-based*), mitte mustripõhine (*pattern-based*) nagu kõikide regulaaravaldiste puhul. Teine erinevus seisneb selles, et funktsioon `CONTAINS` on efektiivsem ja kiirem, kui `LIKE`, sest `CONTAINS` kasutab oma töös täisteksti otsingut (*full text search*). Selle otsingu puhul ei skaneerita tervet rida, vaid kasutatakse teksti indeksit. Indeksi loomisel salvestatakse informatsioon, millises veerus asub väärtus. Näide on toodud päringus 8.

Eelkõige luuakse indeksi:

```
CREATE TEXT INDEX Filmi_Pealkirjad ON Film (Pealkiri);
```

Edasi päritakse informatsioon päringu 10 abil.

Päring 8:

```
SELECT Pealkiri FROM Film CONTAINS(Pealkiri, 'Ra*');
```

- Execution time: 0.016 seconds

Päring näitab filmide pealkirju, mis sisaldavad alamsõna „Ra“. Tuleb olla ettevaatlik ülakomadega, kuna nende kasutamine mõjutab otsingu tulemust. Kui ülakomad on pandud, siis otsitakse täpselt seda sõna või fraasi, mis on ülakomade vahel. Otsingu parameetrite sidumiseks võib kasutada ka `AND` operaatorit. Sellisel juhul otsitakse sõnu või fraase, mis mõlemad esinevad kirjes sõltumata järjekorrast. Sõnade järjekorda arvestatakse ainult ülakomade kasutamisel.

Samas tuleb tähelepanu juhtida, et kui veerule ei ole loodud indeksit, siis tuleb kasutada regulaaravaldist (vt. peatükk 3.3).

3.3 Regulaaravaldised

Nagu programmeerimiskeeltes (näiteks, Java ja Python), on ka SQL-s olemas regulaaravaldised (ing. k. *regular expressions* ehk *regex*), mis võimaldavad teha päringu lühemaks ning tõsta töötlemise kiirust. SQL regulaaravaldised on sisseehitatud märgijadad, mida kasutatakse sõna või teksti otsimiseks. Regulaaravaldised nõuavad täpset teadmist, kuna neid kasutada ja kuidas koostada avaldisi, sest vastasel juhul päringu töötlemine võtab palju aega ning tagastatud tulemus võib erineda oodatud tulemusest. Regulaaravaldisi on mõttekam kasutada väikeste tabelite puhul. Selleks, et regulaaravaldiste kasutamine ei osutuks ebaefektiivseks, tuleb täpselt koostada mustrit, mille järgi hakkab toimuma otsing.

Kaks põhilist regulaaravaldiste operaatorit, mida kasutatakse SQL-s on `SIMILAR TO` ja `REGEXP`. Regulaaravaldised on operaatorite `SIMILAR TO` ja `REGEXP` avaldiste osa, mida pärast võtmesõna `WHERE`. Otsimisel on võimalik ka kasutada `LIKE` operaatorit, mis ei toeta regulaaravaldisi, kuid selle operaatoriga on lubatud kasutada teatuid metamärke. Need on toodud tabelis 3. Operaatorite `SIMILAR TO`, `REGEXP` ja `LIKE` erinevus seisneb otsimismustri erinevas tõlgendamises. Allpool on toodud nende regulaaravaldiste süntaks ning kasutamise näited.

- `SIMILAR TO`

  Antud operaator otsib terve fraasi, kus leidub sisestatud muster. Selle regulaaravaldise süntaks on järgmine:

  `expression [ NOT ] SIMILAR TO pattern [ ESCAPE escape-expression ]`, kus

  `expression` - avaldis, mida on vaja tagastada;

  `pattern` - muster, millele avaldis peab sobima;

  `Escape-expression` - avaldis, mida programm peab ignoreerima kirjetes tulemuse tagastamisel.

Mustrite koostamisel kasutatakse metamärke. Metamärkide kasutamist nimetatakse kärpimiseks. Kärpimine (ingl. k. *truncation*) võimaldab otsida samatüvelisi erineva lõpuga sõnu [9]. Päringus 9 leitakse pealkirju, kus esineb sõne „Rider“.

Päring 9:

```
SELECT * FROM Film WHERE Pealkiri SIMILAR TO 'Rider*';
```

- Execution time: 0.203 seconds

Kui tulemus peab sisaldama, näiteks sõne „Riding“, siis mustri „Rider*“ asemele tuleks panna „Rid*“. Samas kui metamärki * panna sõne algusesse, saadetakse palju ebasobivaid tulemusi. Päringu 10 eesmärk on tagastada kirjete hulka, kus pealkirjas esineb alamsõna „Rio“, kuid tegelik tulemus sisaldab ka pealkirju alamsõnaga „Rid“. Seega tulemuse hulgas on liigsed kirjed, mille tõttu päringu töötlemine võtab rohkem aega.

Päring 10:

```
SELECT * FROM Film WHERE Pealkiri SIMILAR TO 'Ri*';
```

- Execution time: 0.016 seconds

Selleks, et tulemus oleks täpsem ja tagastaks ainult vajalikke kirjeid on mustriks vaja panna „Rio*”. Peale tärni, kärpimiseks saab kasutada kas % märki. Näide on toodud päringus 11.

Päring 11:

```
SELECT Pealkiri FROM Film WHERE Pealkiri SIMILAR TO '12 Angry%';
```

- Execution time: 0.015 seconds

Kui võtta protsendi märk (%) ära, siis otsitakse film, mille pealkiri vastab täpselt ülakomade vahele kirjutatud sõnele (vt. Päring 12).

Päring 12:

```
SELECT Pealkiri FROM Film WHERE Pealkiri SIMILAR TO '12 Angry
Men';
```

- Execution time: 0.016 seconds

Tuleb pöörata tähelepanu, et kärpimine suurendab päringu töötlemise ajakulu, sest metamärgi kasutamisel otsingu ruum suureneb ning tulemuse tagastamiseks tuleb läbi vaadata rohkem

kirjeid. Samuti on kasulik meeles pidada, et protsendi kasutamine on kommutatiivne. Järgmiselt on toodud päringud 13 ja 14, kus on näidatud protsendi märgi kommutatiivsus.

Päring 13:

```
SELECT Pealkiri FROM Film WHERE Pealkiri SIMILAR TO '% Angry Men';
```

- Execution time: 0.015 seconds

Päring 14:

```
SELECT Pealkiri FROM Film WHERE Pealkiri SIMILAR TO '12 % Men';
```

- Execution time: 0.015 seconds

Vaadates päringuid 13 ja 14, võib teha järelduse, et tuleb alati pöörata tähelepanu metamärgi asukohale sõnes, sest see mõjutab tulemust, aga ei mõjuta antud juhul ajakulu. Samuti tasub meeles pidada, et kui tagastatud ridade arv on suur, siis protsendi asukoht mõjutab ka ajakulu.

- `LIKE`

`LIKE` toetab ainult teatud metamärki ning sellega ei saa kasutada keerulist regulaaravaldist. Selle regulaaravaldise operaatori süntaks on:

`expression [ NOT ] LIKE pattern [ ESCAPE escape-expression ]`, kus
`expression` - avaldis, mida on vaja tagastada;
`pattern` - muster, millele avaldis peab sobima.

Selline regulaaravaldis tagastab sama tulemust, mis `SIMILAR TO` operaator. Kasutamise näited koos protsent märgi kommutatiivsusega on toodud päringutes 15, 16 ja 17.

Päring 15:

```
SELECT Pealkiri FROM Film WHERE Pealkiri LIKE '12 % Men';
```

- Execution time: 0.016 seconds

Päring 16:

```
SELECT Pealkiri FROM Film WHERE Pealkiri LIKE '12 Angry %';
```

- Execution time: 0.016 seconds

Päring 17:

```
SELECT Pealkiri FROM Film WHERE Pealkiri LIKE '% Men';
```

- Execution time: 0.015 seconds

Sarnaselt `SIMILAR TO` operaatoriga protsendi asukoht antud juhtudel ei mõjuta ajakulu, aga see on ainult selletõttu, et tagastatud ridade arv on väike. Nii päringutes 13 ja 14, kui ka päringutes 15 ja 16 on tagastatud ridade arv on üks. Päringu 17 tulemusena oli tagastatud 6 rida, kuid ridade arv on siiski väike, et mõjutada ajakulu. Kui ridade arv on tunduvalt suurem (näiteks 500 rida), siis protsendi asukoht hakkab mõjutama päringu täitmiseaega.

`SIMILAR TO` ja `LIKE` operaatori vahe on selles, et `LIKE` muster on lihtne ning lubab kasutamiseks ainult väikest metamärkide hulka, aga `SIMILAR TO` puhul saab kasutada ANSI/ISO SQL standardi regulaaravaldise süntaksit (vt. Tabel 3). Tasub juhtida tähelepanu, et mõlemate regulaaravaldiste puhul andmebaasiserver kasutab võrdsust ja sorteerimist andmete võrdlemiseks.

`LIKE` ja `SIMILAR TO` kasutavad võrdlemist tähtmärkide vahemikku hindamiseks. Tähtmärk on vahemikus, siis kui tema positsioon on võrdne või asub algusmärgi ja lõpumärgi vahemikus. Näiteks, võrdlemine *x* `REGEXP '[A-C]'` on samaväärne, kui *x* `>= A AND` *x* `<= C` [10].

- REGEXP

Selle operaatori süntaks ei ole keerulisem, kui eelmistel regulaaravaldiste operaatoritel. Samas kasutamise võimalusi on rohkem. REGEXP süntaks on toodud allpool:

expression [**NOT**] **REGEXP** *pattern* [**ESCAPE** *escape-expression*],
kus

expression - avaldis, mida on vaja tagastada;
pattern - muster, millele avaldis peab sobima;
ESCAPE escape-expression – muster, mida vaja ignoreerida.

REGEXP toetab sama regulaaravaldiste süntaksi superhulka, mida toetab ka SIMILAR TO. Lisaks REGEXP töötleb erinevalt alakriipsu (_), protsendi (%) ja katuse (^) metamärke. REGEXP käitumine on väga sarnane Perl 5-ga [10]. REGEXP puhul peab muster olema sisestatud väga täpselt, kuna otsitakse 100% sobitamist otsitavas fraasis. See mõjutab ka tõstutundlikkust. Näiteks, 'A' pole sama, kui 'a', kuigi SIMILAR TO ja LIKE operaatorite puhul 'A' peetakse sama väärtusega, kui 'a'.

Allpool toodud tabel näitab vahemiku [A-C] väärtusi REGEXP ning SIMILAR TO ja LIKE operaatorite kasutamisel. Vaadetakse kahte varianti, ühes variandis on andmebaas tõstutundlik (*case-sensitive*) ja teises ei ole.

Tabel 2. Regulaaravaldise väärtuste võrdlemine [10].

	SIMILAR TO, LIKE '[A-C]'	**REGEXP '[A-C]'**
Tõstutundlik	A,B,C,a,b,c,ª,À,Á,Â,Ã,Ä,Å,Æ,Ç,à,á,â,ã,ä,å,æ,ç	A,B,C
Tõstu mittetundlik	A,B,C,b,c,À,Á,Â,Ã,Ä,Å,Æ,Ç,ç	A,B,C

Regulaaravaldise kasutamise näide on toodud päringus 18.

Päring 18:

```
SELECT Pealkiri FROM Film WHERE Pealkiri REGEXP '^Ra.*\s?[0-9]?$';
```

- Execution time: 0.11 seconds

Eelpool toodud päring 18 näitab kõikide filmide pealkirju, mis algavad tähtedega „Ra“ ning mille lõpus võib olla üks number. Päringu 18 tulemus on toodud pildil 10:

	Pealkiri
1	Rambo: First Blood Part II
2	Rambo III
3	Raiders of the Lost Ark
4	Raw Deal
5	Ransom
6	Race the Sun
7	Rapid Fire
8	Rat Race
9	Race You to the Bottom
10	Rambo
11	Ravenous
12	Rain Man
13	Rape Me
14	Raging Bull
15	Rashomon
16	Ratatouille
17	Race with the Devil
18	Rabat
19	Race to Witch Mountain
20	Rainbow Eyes
21	Rabid
22	Rage
23	RagoDnrBg4
24	Ra8hdiNWu2x4Q

Pilt 10. Päringu 18 tulemus.

Tuleb pöörata tähelepanu, et punkt päringus 18 mängib suurt rolli tulemuse kuvamisel. Kuna tegemist on regulaaravaldisega, siis punkt tähendab seda, et pärast sõna algust tuleb veel kas täht, arv või mingi muu sümbol. Tärn pärast punkti ütleb, et neid märke saab (aga ei pruugi) olla mitu tükki järjest.

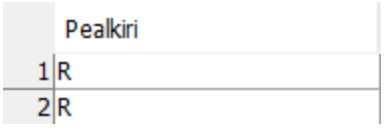
Päringus 19 on toodud näide päringust, kus on kasutatud regulaaravaldist ning punkt jääb mustrisse panemata.

Päring 19:

```
SELECT Pealkiri FROM Film WHERE Pealkiri REGEXP '^Ra*\s?[0-9]?$';
```

- Execution time: 0.22 seconds

Päringu 19 tulemus on toodud allpool pildil 11:



	Pealkiri
1	R
2	R

Pilt 11. Päringu 19 tulemus.

Allpool on toodud kokkuvõtlik tabel metamärkide kasutamise kohta erinevate regulaaravaldiste puhul, mis aitab aru saada metamärgi tähendust erinevate regulaaravaldise operaatorite puhul.

Tabel 3. Metamärkide kasutamine erinevates regulaaravaldistes [10].

Märk	SIMILAR TO	LIKE	REGEXP	Tähendus
%	+	+	+	sümbolite sobimine
*	-	-	+	sümboli(te) sobimine kas 0 või rohkem korda
.	+	-	+	`REGEXP` puhul ühe sümboli sobimine; `SIMILAR TO`– vahemiku tähendamine
-	+	+	+	vahemiku määramine
+	-	-	+	sümboli(te) sobimine kas 1 või rohkem korda
?	-	-	+	sümbolite sobimine kas 0 või 1 kord
$	+	-	+	sõna lõpp
|	-	-	+	võimalike alternatiivide määramine
_	-	+	+	ühe sümboli sobimine
\	+	-	+	`ESCAPE` funktsiooni ekvivalent
^	+	+	+	`REGEXP` puhul – kui ^ on pandud enne sümbolite vahemiku, siis määrab sõna algust, kui ^ on asub vahemikus, siis kõik sümboleid peale neid, mis on vahemiku sees; `SIMILAR TO` ja `LIKE` puhul - ekvivalentne lahutamise operaatoriga;
:	+	-	+	alamklassi määramine
[and]	+	+	+	sümbolite vahemiku määramine
(and)	-	-	+	regulaaravaldise osade rühmitamine
{and}	-	-	+	koguse määramine

3.4 CONTAINS, SIMILAR TO, LIKE ja REGEXP operaatorite võrdlus

Tabelis 4 on toodud neli päringut operaatorite tööd võrdlemiseks. Kuigi päringud on erinevad, tulemus on sama.

Tabel 4. CONTAINS, SIMILAR TO, LIKE ja REGEXP operaatorite võrdlus.

Rea number	Operaator	Aeg (sec)	Puu	Tagastatud ridade arv
1.	`SELECT Pealkiri FROM Film CONTAINS(Pealkiri,'%Angry%');`	0.016		2
2.	`SELECT Pealkiri FROM Film WHERE Pealkiri SIMILAR TO '%Angry%';`	0.02		2
3.	`SELECT Pealkiri FROM Film WHERE Pealkiri LIKE '%Angry%';`	0.016		2
4.	`SELECT Pealkiri FROM Film WHERE Pealkiri REGEXP '.*Angry.*';`	0.012		2

Analüüsides tabeli 4, võib teha järelduse, et operaator `LIKE` on kõige ebaefektiivsem ning operaator `REGEXP` kõige kiirem. Kui võrrelda nende ajakulu, siis tuleb välja, et `REGEXP` töötab 0.016 / 0.02 = 0.8 korda kiiremini. Nagu näha `CONTAINS`, `LIKE` ja `SIMILAR TO` puhul olid kasutatud samad regulaaravaldised. Kuna `REGEXP` ei toeta protsendi märgi kasutamist, siis `REGEXP` operaatori jaoks oli koostatud sama põhimõttega regulaaravaldis, kasutades punkti ja tärni. Kui võrrelda päringute puid, siis need on samasugused.

3.5 TOP ja WHERE piirangud päringutes

Mõnedes olukordades tulemuse saamiseks pole vaja vaadata tabelis kõiki andmeid. Kasutades otsingu parameetreid, saab piirata otsingu ruumi ja vähendada päringu ajakulu. Allpool päringus 20 on toodud näide otsingu parameetrite kasutamisest, mis näitab ekraanile kõik filmid, mille ilmumise aasta on 2011.

Päring 20:

```
SELECT * FROM Film WHERE Aasta=2011;
```

- Execution time: 0.032 seconds

Kuna päringu 20 tulemus on mahukas (75 rida), siis pildil 12 on toodud osa tulemusest :

	id	Pealkiri	Aasta	Riik	Kestus	Keel	Žanr	Režissöör	Eelarve
1	143	Mission: Impossible - Ghost Protocol	2 011	Ameerika	142	saksa	draama	110	330 396
2	296	Scream 4	2 011	Korea	134	itaalia	melodraama	137	952 349
3	320	X-Men: First Class	2 011	Argentina	97	itaalia	sõjafilm	91	567 160
4	506	Spy Kids: All the Time in the World in 4D	2 011	Belgia	170	itaalia	perefilm	95	979 253
5	706	Wrong Turn 4: Bloody Beginnings	2 011	Hispaania	110	saksa	sõjafilm	112	742 324
6	833	Final Destination 5	2 011	Itaalia	99	inglise	seiklus	34	208 264
7	970	Hostel: Part III	2 011	Korea	90	hispaania	sõjafilm	155	775 180
8	1 302	Transformers: Dark of the Moon	2 011	Hiina	161	saksa	kriminaal	151	998 127
9	2 168	Sanctum	2 011	Hiina	96	itaalia	triller	46	886 737
10	2 169	Battle Los Angeles	2 011	Brasilia	162	itaalia	melodraama	46	993 276

Pilt 12. Päringu 20 tulemus.

Päringu täitmisel kontrollib *SQL Anywhere* iga kirjet andme tabelist. Päringu kasutamine on ebaefektiivne juhul, kui tagastatud ridade arv on väiksem kui 2% kõikide ridade arvust antud tabelis. Päringu efektiivsust saab suurendada indeksite abil, mille kasutamine on detailsemalt kirjeldatud

neljandas peatükis. Päring 21 vaatab kogu tabeli üle ning otsib kõik filmid, mille ilmumise aasta on 1995 ning mis on võitnud Oscarit.

Päring 21:

```
SELECT Oscar.id as Oscari_id , Film.id as Filmi_id, Film.Pealkiri as
Pealkiri FROM Oscar JOIN Film on Film.id = Oscar.Film WHERE
Film.Aasta = 1995 ORDER BY Film.Pealkiri;
```

- Execution time: 0.078 seconds

Juhul kui on vaja kontrollida ainult andmete olemasolu tabelis, siis tuleb parandada päringut, lisades `TOP` operaatorit [9]. `TOP` operaator piirab tagastatavate ridade arvu etteantud väärtusega. Arvu tüüp peab olema 32-*bit unsigned integer* vahemikus 0-st kuni 2^{32}-1 (4GB-1 või 4,294,967,295), kus 0 tähendab, et päring ei tagasta ühtegi rida. Näide on toodud päringus 22, mis vaatab, kas tabelis on vähemalt üks Oscari film, mille ilmumise aasta on 1995.

Päring 22:

```
SELECT TOP 1 Oscar.id as Oscari_id , Film.id as Filmi_id,
Film.Pealkiri as Pealkiri FROM Oscar JOIN Film on Film.id =
Oscar.Film WHERE Film.Aasta = 1995 ORDER BY Film.Pealkiri;
```

- Execution time: 0.014 seconds

Päringu 22 tulemus on toodud pildil 13:

	Oscari_id	Filmi_id	Pealkiri
1	5 116	225	Ace Ventura: When Nature Calls

Pilt 13. Päringu 22 tulemus.

Sõltuvalt päringu eesmärgist, `TOP` operaatorit kasutamine vähendab tunduvalt päringu töötlemise aega. Võrreldes päringuid 21 ja 22, võib järeldada, et `TOP` operaator kiirendab otsingut 0.078 / 0.014 = 5.5 korda. Seega, kui on vaja kontrollida andmete olemasolu andmebaasis, siis `TOP` operaator on hea abimees. Tuleb aga meeles pidada, et `TOP` eeldab andmete sorteerimist siis, kui veerule pole määratud unikaalse omadusega kitsendust või võtit.

3.6 Päringute restruktureerimine

Mõnedel juhtudel SQL päringu ümberkirjutamine oluliselt mõjutab päringu efektiivsust. Kui kasutaja saab päringu eesmärgist aru, siis ta oskab päringut muuta nii, et päring tagastaks nõutud tulemust, kasutades selleks võimalikult vähem aega. Allpool on toodud põhilised vihjed [12], millest alustada päringu modifitseerimist:

1. Komponeeri **predikaate**, kasutades `AND` ja võrdusmärgki (`=`).
   Näiteks,

```
SELECT PEALKIRI FROM FILM AS F JOIN REŽISSÖÖR AS R ON
F.REŽISSÖÖR = R.ID WHERE R.EESNIMI = 'Anna' AND R.PERENIMI =
'Foerster';
```

2. Sõnasta otsingu tingimused võimalikult lihtsalt ja lühidalt. Väldi keerulist sõnastust.
3. Ära pane palju funktsioone ja protseduure ühte SQL päringusse. See mõjutab negatiivselt päringu efektiivsust ning päringu töötlemine võtab rohkem aega, kui mitme ühendatud väiksemate SQL päringute käivitamine.

Mida lühem on päring ning mida lihtsam on selle töötamise loogika, seda lihtsam on päringust aru saada ning vastavalt vajadusele muuta. Lihtsus teeb päringut efektiivsemaks.

3.7 Alampäringute koostamine

Kvantitatiivseid predikaate (nagu `ANY`, `ALL`, `EXISTS` või `IN`) kasutatakse alampäringutes koos `WHERE` ja `HAVING` operaatoritega. Selliste predikaatidega alampäringud tagastavad väärtuste hulka, mida edaspidi kasutatakse tulemuse saamiseks. Predikaadid jaguvad kolmeks tüübiks [6]:

- **ANY/ALL**

  Need operaatorid mõjutavad võrdlemist ning nende kasutamiseks peab olema alampäringus sellised laused nagu `GROUP BY` või `HAVING`. Süntaks on toodud allpool:

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE expression2 comparison_operator {ALL | ANY | SOME} (
subquery ) , kus
```

`column_name` – veeru nimi, mille kohta tehakse päringut;

`expression1` – avaldis, mida peab tagastama;

`table_name` – tabeli nimi, kus asub kasutatav veerg;

`expression2` – avaldist, mille vastu rakendatakse võrdlus operaatorit;

`comparison_operator` – võrdlus operaator (=, <>, !=, >, >=, <, või <=);

`subquery` - alampäring

`ANY` operaator tagastab tulemust siis, kui vähemalt üks etteantud tingimustest kehtib. `ALL` operaator tagastab kirjet siis, kui kõik tingimused on täidetud. Päringus 23 on toodud `ANY` operaatori kasutamise näide.

Päring 23:

```
SELECT Pealkiri, id FROM Film
WHERE id = ANY
(SELECT Film FROM Oscar WHERE Film > 50 AND Pealkiri REGEXP
'12.*') GROUP BY id, Pealkiri;
```

- Execution time: 0.062 seconds

Päring 23 tagastab filmide pealkirju ja id-sid, kui filmi pealkiri algab arvuga 12, on võitnud Oscari ning filmi indeks on suurem kui 50. Operaatori `ANY` alternatiivina saab kasutada operaatorit `SOME`. Sama tulemus saab kätte ka `IN` operaatori kasutamisel. `IN` operaatori kasutamist näidetakse päringus 25. Päringu 23 tulemus on toodud pildil 14:

	Pealkiri	id
1	12:01	240

Pilt 14. Päringu 23 tulemus.

Päringus 24 on toodud päringuga 23 sarnane päring, kus `ANY` operaatori asemel kasutatakse operaatorit `ALL`.

Päring 24:

```
SELECT Pealkiri FROM Film
WHERE id = ALL
(SELECT Film FROM Oscar WHERE Film > 50 AND Pealkiri REGEXP
'12.*');
```

- Execution time: 0.018 seconds

Kuna päringu 24 tulemus on väga mahukas (9998 rida), siis pildil 15 on toodud ainult osa tulemusest:

	Pealkiri
1	Fleshburn
2	First Blood
3	Rambo: First Blood Part II
4	Rambo III
5	Raiders of the Lost Ark
6	Beethoven
7	Shinkansen daibakuha
8	E.T. the Extra-Terrestrial
9	Terminator 2: Judgment Day
10	The Terminator

Pilt 15. Päringu 24 tulemus.

Võib tähele panna, et päringu 24 tulemusel on toodud filmide pealkirjad, mille pealkiri ei alga arvuga 12. See tähendab, et teine tingimus ei kehti. Kuid kehtib tingimus, kus filmi ID number on suurem, kui 50, mille tõttu `ALL` operaator loeb kirje tõeseks ja näitab ekraanile.

- **IN/NOT IN**

Need operaatorid annavad võimaluse panna mitu väärtust tingimuses. Näide on toodud päringus 25.

Päring 25:

```
SELECT Eesnimi + ' ' + Perenimi  As Nimi FROM Režissöör
WHERE Eesinimi  IN ('Anna', 'Robert');
```

- Execution time: 0 seconds

Päringu 25 tulemus on toodud pildil 16:

	Nimi
1	Robert Zemeckis
2	Robert Altman
3	Robert Wise
4	Robert Rodriguez
5	Anna Foerster

Pilt 16. Päringu 25 tulemus.

SQL päringu süntaks nende operaatorite kasutamisega on toodud allpool:

```
SELECT [column_name... | expression ]
FROM [table_name]
{WHERE | HAVING | {AND | OR}} value [NOT] IN ({comp_value1,
comp_value2[, ...] | subquery}); , kus
```

`column_name` – veeru nimi, mille kohta käivitatakse päring ning mille andmeid tagastatakse;

`expression` – avaldis, mida tagastatakse päringu tulemusena;

`table_name` – tabeli nimi, kus käivitatakse päringut;

`{WHERE | HAVING | {AND | OR}} value` – tingimused, mis peavad kehtima tulemuse tagastamiseks;

`NOT` – operaator väärtuste välistamiseks;

`comp_value1, comp_value2...| subquery` – väärtused või alampäring;

Teine võimalus `IN` ja `NOT IN` operaatorite kasutamiseks on lisada tingimuseks alampäringusse. Näide on toodud päringus 26.

Päring 26:

```
SELECT Eesnimi + ' ' + Perenimi  As Nimi FROM Režissöör
WHERE id  IN
(SELECT Režissöör FROM Film JOIN Oscar ON Oscar.Film = Film.id);
```

- Execution time: 0.047 seconds

Päring 26 tagastab režissööride nimed, kelle film on võitnud Oscari. Kuna päringu 26 tulemus on mahukas (245 rida), siis pildil 17 on toodud ainult osa tulemusest:

	Nimi
1	Zack Snyder
2	Mel Gibson
3	Luis Bu¤uel
4	Krzysztof Kieslowski
5	Jean-Luc Godard
6	Tom Hooper
7	William Friedkin
8	uMsNfRmkNMdxx8 BCm
9	CfMrqrSMId 8NunxusnU4NIGIUeIIXP1MrDIcI
10	UzcTFmx12YLjnELxB9VITjaBN1 5nrzLgJN4rQz4bf

Pilt 17. Päringu 26 tulemus.

Päring 27 on päringu 25 vastase eesmärgiga: päring 27 näitab režissööride nimesid, kelle film ei võitnud Oscarit.

Päring 27:

```
SELECT Eesnimi + ' ' + Perenimi  As Nimi FROM Režissöör
WHERE id  NOT IN
(SELECT Režissöör FROM Film JOIN Oscar ON Oscar.Film = Film.id);
```

- Execution time: 0.015 seconds

Kuna päringu 27 tulemus on väga mahukas (9756 rida), siis pildil 18 on näidatud ainult osa tulemusest.

	Nimi
1	Sergio Leone
2	U 3IVFie3WJkc8yJ
3	TommcTruNMLHep0i4LOk FgY28NMFwQxmbyHpqvPujTibo
4	C9kW0Au5c lncDfzG5fznoH42cfNAOXDI6s
5	9KdV uIcv9dQcveiX6NjYJzsMB
6	VbK9Xmsy5LA9Vjpb QR
7	87O1eH1h1k44 hbjCwWFTSjS1yIIn
8	n9JTuEWXkWHdTvGejf zXRAF66bMVeNatHHZCXOHi04
9	Gsu5n mDwPo2NZa
10	8eKJJaemr2C9TVwQcLe0W H7slykmv1O6aBGK

Pilt 18. Päringu 27 tulemus.

- **EXISTS/NOT EXISTS**

`EXISTS` operaator kontrollib, kas alampäringu tulemus on olemas või alampäring tagastab tühja hulga. Süntaks on toodud allpool:

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE [NOT] EXISTS (subquery), kus
```

`column_name` – veeru nimi, mille kohta käivitatakse päring ning mille andmeid tagastatakse;

`expression1` – avaldis, mida tagastatakse päringu tulemusena;

`table_name` – tabeli nimi, kus käivitatakse päring;

`WHERE [NOT] EXISTS subquery` – alampäring, mis kontrollib, kas rida vastab tingimustele või mitte.

Päring 28 on päringuga 26 sarnane, kuid `IN` asemel kasutatakse `EXISTS`.

Päring 28:

```
SELECT Eesnimi + ' ' + Perenimi  As Nimi FROM Režissöör
WHERE  EXISTS
(SELECT Režissöör FROM Film JOIN Oscar ON Oscar.Film = Film.id);
```

- Execution time: 0.006 seconds

Kuna päringu 28 tulemus on väga mahukas (9999 rida), siis pildil 19 on toodud ainult osa tulemusest:

	Nimi
1	Steven Spielberg
2	Martin Scorsese
3	Alfred Hitchcock
4	Stanley Kubrick
5	Quentin Tarantino
6	Orson Welles
7	Francis Ford Coppola
8	Ridley Scott
9	Akira Kurosawa
10	Joel Coen
11	John Ford
12	Sergio Leone

Pilt 19. Päringu 28 tulemus.

Päring 29 on päringu 28 vastase eesmärgiga. Sarnaselt päringule 27, näitab see režissööre, kelle film ei võitnud Oscarit.

Päring 29:

```
SELECT Eesnimi + ' ' + Perenimi  As Nimi FROM Režissöör
WHERE  NOT EXISTS
(SELECT Režissöör FROM Film JOIN Oscar ON Oscar.Film = Film.id);
```

- Execution time: 0.001 seconds

Päring 29 tagastas null rida:

Pilt 20. Päringu 29 tulemus.

Kahes järgmises peatükis võrreldakse operaatoreid `IN` ja `EXISTS` ning seletatakse, mis olukorras ja millist operaatorit on parem kasutada.

3.7.1 EXISTS ja IN alampäringutes

Nende operaatorite kasutamise reeglid pole absoluutsed. Teatud tingimustes on parem kasutada `IN`, kui `EXISTS`, ning vastupidi. Päringu koostamisel tuleb lähtuda järgmisest soovitusest [12]:

`EXISTS` on palju kiirem, kui alampäringu tulemus väga mahukas ning `IN` on palju kiirem, kui alampäringu tulemus on väike.

See väide on rohkem rakendatav **OLTP** keskkondades, kus juurdepääs alam- ja kõrvalpäringutele käib läbi indekseeritud veeru kõrge valikulisusega. **DSS** keskkondades saab olla madal valikulisus alam- ja kõrvalpäringutes ning ühendatud veergudes ei pea olema ühtegi indeksit. DSS keskkondades on kombeks saanud kasutada `EXISTS` operaatorit. Allpool on võrreldud kahte päringut. Esimene (Päring

30) näitab `EXISTS` operaatori kasutamist alampäringus ning teine (Päring 31) demonstreerib `IN` operaatori kasutamist alampäringus.

Päring 30. `EXISTS` operaatori kasutamine alampäringus:

```
SELECT TOP 1000 Režissöör.id, Režissöör.Eesnimi + ' ' +
Režissöör.Perenimi AS Nimi, Režissöör.Sugu,Režissöör.Rahvus
FROM Režissöör WHERE EXISTS
(SELECT Režissöör.id FROM Film INNER JOIN Vaatamine ON Film.ID =
Vaatamine.Film INNER JOIN Režissöör ON Režissöör.id = Film.Režissöör
WHERE Režissöör.id = Film.Režissöör AND Vaatamine.Hinne > 800 ) ORDER
BY Nimi;
```

- Execution time: 0.031 seconds

Antud päring näitab režissööride andmeid, kelle filmi hinne on suurem kui 800. Kuna päringu 30 tulemus on mahukas (1000 rida), siis pildil 21 on toodud ainult osa tulemusest.

	id	Nimi	Sugu	Rahvus
1	8 085	0 d8zrsu89OxNpDd85agFQf	F	YH7
2	6 604	0 Emz1O	F	v5L
3	4 643	0 fF9n8ah	M	OV
4	594	0 gfJymnKYsIC9iiX1	M	z6F3OzzjdoFUI
5	9 446	00Vgv60snpM1Ol wdVQVKAOt3zH90HzrbR	F	6QYOPkBS7A8Uq8zYrav
6	8 641	01qyrl RpWpfcOxtCvEEny5Z711vtw	M	u19WO8F9IN3wJq30E8Z9L
7	1 444	02A8q 9m8ikzt8wTVE3ndHJNEk	M	J6S4xYMdxsQTMzvBatBDJCQx4
8	2 235	02PqTfLdrtQGqSgR1fr5P0eJO3 mUnfNPWKltXPDACm417R2OTe1	M	ZUDX2XPb
9	2 629	02uhKgYrQ7uaz9xrqwVl5 2I5Xgdk	F	jn2ficbNhDyBE5s8
10	8 473	031m3obyp BuIvF	F	81kBYNCtj4qEc

Pilt 21. Päringu 30 osa tulemusest.

Päringu 30 alampäring sisaldab 2011 rida. Päring 31 näitab, kuidas `IN` kasutamine saab tõsta produktiivsust.

Päring 31:

```
SELECT TOP 1000 Režissöör.id, Režissöör.Eesnimi + ' ' +
Režissöör.Perenimi AS Nimi , Režissöör.Sugu, Režissöör.Rahvus
FROM Režissöör WHERE Režissöör.id IN
(SELECT Režissöör.id FROM Film
INNER JOIN Vaatamine ON Film.ID = Vaatamine.Film
WHERE Vaatamine.Hinne > 800 )ORDER BY Nimi;
```

- Execution time: 0.016 seconds

Päringu 31 alampäring sisaldab 2011 rida. Kuna päringu 31 tulemus on mahukas (1000 rida), siis pildil 22 on toodud osa tulemusest.

	id	Nimi	Sugu	Rahvus
1	8 085	0 d8zrsu89OxNpD...	F	YH7
2	6 604	0 Emz1O	F	v5L
3	4 643	0 fF9n8ah	M	OV
4	594	0 gfJymnKYsIC9iiX1	M	z6F3Ozzjd...
5	9 446	00Vgv60snpM1Ol ...	F	6QYOPkB...
6	8 641	01qyrl RpWpfcOxt...	M	u19WQ8F...
7	1 444	02A8q 9m8ikzt8w...	M	J6S4xYMd...
8	2 235	02PqTfLdrtQGqSg...	M	ZUDX2XPb
9	2 629	02uhKgYrQ7uaz9x...	F	jn2ficbNh...
10	8 473	031m3obyp BuIvF	F	81kBYNCt...

Pilt 22. Päringu 31 tulemus.

On näha, et päring 31, mis kasutab `IN` operaatorit, töötab tunduvalt kiiremini, kui päring 30. Selleks on kaks põhjust. Esimene on see, et alapäringu tulemus ei ole mahukas (umbes 2000 rida) ning teine põhjus on see, et võrdlemiseks on kasutatud staatiline hulk andmeid. Tulemus on saadud kätte 0.031 / 0.016 = 1.9 korda kiiremini. Järgmised kaks päringut 32 ja 33 aitavad võrrelda `EXISTS` ja `IN` tööd, kui alampäringu tulemus on mahukas.

Päring 32:

```
SELECT TOP 1000 Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi
FROM Režissöör
WHERE Režissöör.Sünniaeg > '1950-01-01'
AND Režissöör.Rahvus = 'ameeriklane'
AND Režissöör.ID IN (SELECT Film.Režissöör FROM FILM WHERE
LEN(Film.Pealkiri) < 15) ORDER BY Nimi;
```

- Execution time: 0.016 seconds

Päring näitab režissööride nimesid, kes on sündinud hiljem, kui 1. Jaanuar aastal 1950, kelle rahvus on „ameeriklane“ ning kelle filmi pealkirja pikkus vähem, kui 15 sümbolit. Päringu tulemuses on 1000 rida ning alampäring sisaldab 5387 rida. Ehk päringute 34 ja 35 alampäringud on 5387 / 2011 = 2.67 korda suurem, kui päringute 30 ja 31 alampäringute tulemus. Kuna päringu 32 tulemus on mahukas (1000 rida), siis pildil 23 on toodud ainult osa tulemusest.

	Nimi
1	60D rmYvAocYm8DBc6eQtPtMU
2	Alfonso Cuarçn
3	Anna Foerster
4	Buster Keaton
5	C LBotO6DWu
6	DFI9A3tEXec0PEaJKsHL1rO s
7	ezt1ohVFY4ovtwN1CW6Luq90h aDyPcxrR
8	f0KVovFYf8s8u4Ol6bbm CqjiNvDF
9	Federico Fellini
10	fgk33I7ABrCoscLKmKE5Bn25y 0OfsnRnR2bz

Pilt 23. Päringu 32 tulemus.

Allpool on toodud sama tulemusega päring 33, mis kasutab `EXISTS` operaatorit.

Päring 33:

```
SELECT TOP 1000 Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi
FROM Režissöör
WHERE Režissöör.Sünniaeg > '1950-01-01' AND Režissöör.Rahvus =
'ameeriklane'
AND EXISTS (SELECT Režissöör FROM Film WHERE LEN(Film.Pealkiri) < 15)
ORDER BY Nimi;
```

- Execution time: 0.010 seconds

Päringute 32 ja 33 tulemused on samad. Sarnaselt päringu 34 tulemusega, päringu 33 tulemus on mahukas (1000 rida), seega pildil 24 on toodud päringu 33 osa tulemusest.

	Nimi
1	60D rmYvAocYm8DBc6eQtPtMU
2	Alfonso Cuarçn
3	Anna Foerster
4	Buster Keaton
5	C LBotO6DWu
6	DFI9A3tEXec0PEaJKsHL1rO s
7	ezt1ohVFY4ovtwN1CW6Luq90h aDyPcxrR
8	f0KVovFYf8s8u4Ol6bbm CqjiNvDF
9	Federico Fellini
10	fgk33I7ABrCoscLKmKE5Bn25y 0OfsnRnR2bz

Pilt 24. Päringu 33 tulemus.

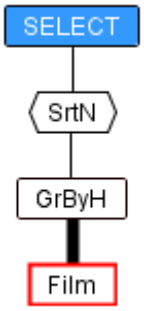
Võrreldes kahte päringu täitmise aega, on näha, et antud juhul päring operaatoriga `EXISTS` töötab 0.016 / 0.010 = 1.6 korda kiiremini. Kokkuvõttes, saab teha järelduse, et `IN` operaator on õige valik siis,

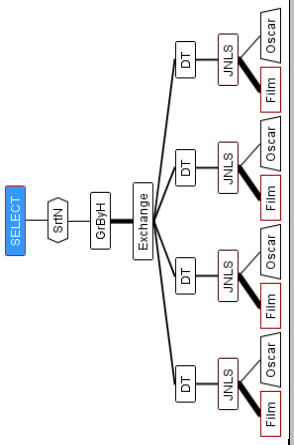
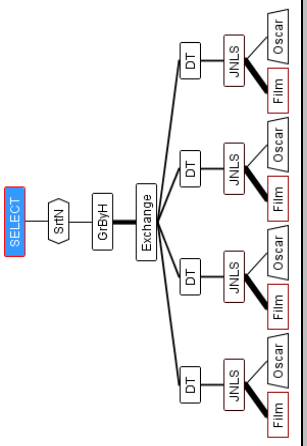
kui alampäringu tulemus ei ole väga mahukas. Kui alampäring on mahukas, siis tasub kasutada `EXISTS` operaatorit.

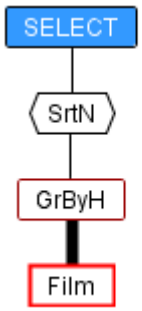
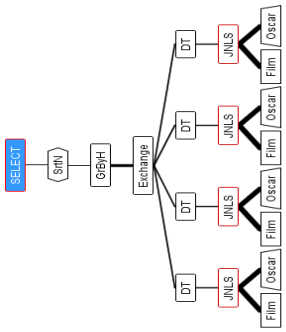
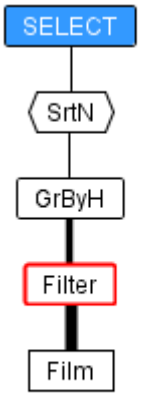
3.7.2 Alampäringute operaatorite võrdlus

Tabelis 5 on toodud päringud, mis võrdlevad päringuid erinevate operaatoritega. Iga päring tagastab 500 filmi pealkirja. Alampäring (`SELECT DISTINCT Oscar.Film FROM Oscar JOIN Film ON Film.id = Oscar.Film WHERE Film.Keel = 'inglise'`) tagastab 951 rida.

Tabel 5. Alampäringute operaatorite võrdlus.

Rea number	**Operaator**	**Aeg (sec)**	**Puu**	**Päring**
1.	ALL	0.015		SELECT TOP 500 Pealkiri FROM Film WHERE Film.id = ALL (SELECT DISTINCT Oscar.Film FROM Oscar JOIN Film ON Film.id = Oscar.Film WHERE Film.Keel = 'inglise') GROUP BY Pealkiri ORDER BY Pealkiri;

2.	ANY	0.016		SELECT TOP 500 Pealkiri FROM Film WHERE Film.id = ANY (SELECT DISTINCT Oscar.Film FROM Oscar JOIN Film ON Film.id = Oscar.Film WHERE Film.Keel = 'inglise') GROUP BY Pealkiri ORDER BY Pealkiri;
3.	IN	0.015		SELECT TOP 500 Pealkiri FROM Film WHERE Film.id IN (SELECT DISTINCT Oscar.Film FROM Oscar JOIN Film ON Film.id = Oscar.Film WHERE Film.Keel = 'inglise') GROUP BY Pealkiri ORDER BY Pealkiri;

4.	NOT IN	0.016		SELECT TOP 500 Pealkiri FROM Film WHERE Film.id NOT IN (SELECT DISTINCT Oscar.Film FROM Oscar JOIN Film ON Film.id = Oscar.Film WHERE Film.Keel = 'inglise') GROUP BY Pealkiri ORDER BY Pealkiri;
5.	EXISTS	0.016		SELECT TOP 500 Pealkiri FROM Film WHERE EXISTS (SELECT DISTINCT Oscar.Film FROM Oscar WHERE Film.Keel = 'inglise') GROUP BY Pealkiri ORDER BY Pealkiri;
6.	NOT EXISTS	0.063		SELECT TOP 500 Pealkiri FROM Film WHERE NOT EXISTS (SELECT DISTINCT Oscar.Film FROM Oscar WHERE Film.Keel = 'inglise') GROUP BY Pealkiri ORDER BY Pealkiri;

Võrreldes omavahel operaatorite IN, ANY ja ALL ajakulu, järeldub, et ANY operaator on tunduvalt aeglasem, kui teised. Selle põhjus on see, et ANY operaatori puhul peavad kõik etteantud tingimused kehtima, mille kontrollimiseks kulub aega.

Vaadates operaatorite EXISTS ja NOT EXISTS ajakulu, tuleb tähele panna, et NOT EXISTS töötab 0.063 /0.016 = 3.9 korda aeglasemalt, kui EXISTS operaator.

Analüüsides päringuid operaatoritega IN ja EXISTS, siis on näha, et päringut operaatoriga IN täidetakse 0.015 sekundiga ning operaatoriga EXISTS - 0.016 sekundiga. See tõestab, et kui alampäring ei ole väga mahukas (antud juhul vähem kui 1 000 rida), siis IN operaator töötab efektiivsemalt.

3.8 JOIN lause

JOIN lauset kasutatakse selleks, et ühendada andmeid erinevatest tabelitest ühte hulka. Tavaliselt ühendatakse paaride kaupa, aga on võimalik kasutada JOIN mitu korda järjest ühes päringus. Kui koostatud päring on liiga keeruline, siis kompilaator ise loob ajutised ühendused. On olemas kaks JOIN lause süntaksi tüüpi: *Comma-Separated* süntaks ning *ANSI* süntaks [13]. Nendest räägitakse täpsemini edasi punktides 3.8.1 ja 3.8.2.

3.8.1 *ANSI* süntaks

ANSI süntaksiga JOIN lauset on mitu liiki, mis võib jagada omavahel kaheks rühmaks – INNER JOIN ja OUTER JOIN. Kui INNER JOIN tagastab ainult read, mis on olemas mõlemates tabelites, siis OUTER JOIN tagastab kõik read ühest tabelist (sõltuvalt mis JOIN liiki kasutada) ning read teisest tabelist, mis ühtivad etteantud tingimustega. Edasi vaadetakse igat liiki täpsemalt [14]:

- **INNER JOIN**

  Antud tüüp tagastab need andmed, mis klapivad omavahel kokku nii vasakus, kui ka paremas tabelis. Näide on toodud päringus 34.

Päring 34:

```
SELECT Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör INNER JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05';
```

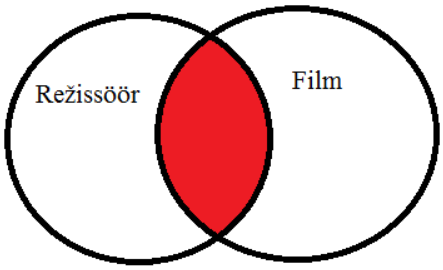
- Execution time: 0.031 seconds

Kuna tulemus on väga mahukas (8645 rida), pildil 25 on toodud osa tulemusest:

	id	Nimi	Režissöör	Pealkiri
1	176	GCNPIOAHT6MBew E6LKgBFtvIS6pXxkExEcyz2dU	176	Maximum Risk
2	65	Ron Howard	65	Drive
3	120	791 xo95nq3i3dbJUD	120	Jaws 2
4	101	William Friedkin	101	Jaws 3-D
5	58	Elia Kazan	58	Evil Dead II
6	94	Robert Rodriguez	94	Army of Darkness
7	207	NZTG0tJXXXYfl pcB	207	Piranha
8	106	KFt wVtvkAOJJX517Aq49b634	106	Missing in Action 2: The Beginning
9	200	fqia ioVsgk8	200	Braddock: Missing in Action III
10	60	Vincente Minnelli	60	Speed 2: Cruise Control

Pilt 25. INNER JOIN lause kasutamise näide.

Teiste sõnadega `INNER JOIN` annab tabelite „Režissöör“ ja „Film“ ühisosa, mis on näidatud pildil 26:



Pilt 26. Tabeli „Režissöör“ ja „Film“ ühisosa.

- **`LEFT JOIN`**

`LEFT JOIN` tagastab kõik read vasakust tabelist, sõltumata sellest kas read klappivad etteantud tingimustega või mitte ning need read paremast tabelist, mis klappivad etteantud tingimustega. Näide on toodud päringus 35:

Päring 35:

```
SELECT Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör LEFT JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05';
```

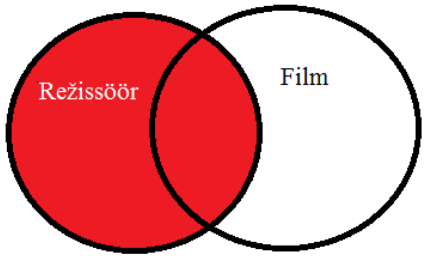
- Execution time: 0.022 seconds

Kuna tulemus on väga mahukas (17645 rida), allpool on toodud osa tulemusest pildil 27:

	id	Nimi	Režissöör	Pealkiri
1	176	GCNPIOAHT6MBew E6LKgBFtvIS6pXxkExEcyz2dU	176	Maximum Risk
2	65	Ron Howard	65	Drive
3	120	791 xo95nq3i3dbJUD	120	Jaws 2
4	101	William Friedkin	101	Jaws 3-D
5	58	Elia Kazan	58	Evil Dead II
6	94	Robert Rodriguez	94	Army of Darkness
7	207	NZTG0tJXXXYfl pcB	207	Piranha
8	106	KFt wVtvkAOJJX517Aq49b634	106	Missing in Action 2: The Beginning
9	200	fqia ioVsgk8	200	Braddock: Missing in Action III
10	60	Vincente Minnelli	60	Speed 2: Cruise Control

Pilt 27. LEFT JOIN lause kasutamise näide.

Visuaalne `LEFT JOIN` tulemus on toodud pildil 28.



Pilt 28. LEFT JOIN visuaalne tulemus.

- **`RIGHT JOIN`**

  `RIGHT JOIN` tagastab kõik read paremast tabelist, sõltumata sellest kas read klappivad etteantud tingimustega või mitte ning need read vasakust tabelist, mis klappivad etteantud tingimustega. Näide on toodud päringus 36:

Päring 36:

```
SELECT Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör RIGHT JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05';
```

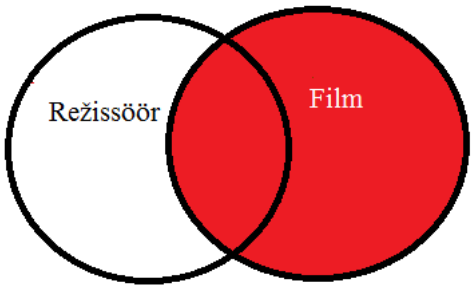
- Execution time: 0.022 seconds

Kuna tulemus on väga mahukas (8645 rida), allpool on toodud osa tulemusest (Pilt 29):

	id	Nimi	Režissöör	Pealkiri
1	176	GCNPIOAHT6MBew E6LKgBFtvIS6pXxkExEcyz2dU	176	Fleshburn
2	68	Tony Scott	68	First Blood
3	63	Kathryn Bigelow	63	Rambo: First Blood Part II
4	49	Michael Curtiz	49	Rambo III
5	98	Krzysztof Kieslowski	98	Raiders of the Lost Ark
6	74	Blake Edwards	74	The Terminator
7	231	CQ2FzBA3mWKMCRxOpxpNw4VaXP scekOu5FaWgGF	231	Terminator 3: Rise of the Machines
8	43	Werner Herzog	43	Terminator Salvation
9	21	David Lynch	21	Commando
10	26	Tim Burton	26	Raw Deal

Pilt 29. RIGHT JOIN lause kasutamise näide.

Visuaalne `RIGHT JOIN` tulemus on toodud pildil 30.



Pilt 30. RIGHT JOIN visuaalne tulemus.

- **`FULL OUTER JOIN`**

  Antud `JOIN` tüüp tagastab andmeid sõltumata sellest, kas väärtused klappivad teise tabeli andmetega. Näide on toodud päringus 37.

Päring 37:

```
SELECT Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör FULL OUTER JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05';
```

- Execution time: 0.021 seconds

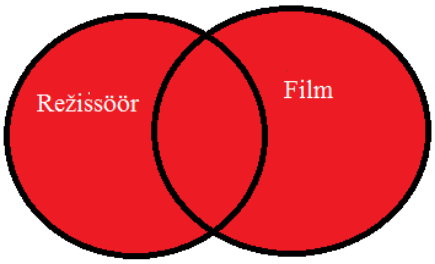
Kuna tulemus on väga mahukas (17645 rida), allpool on toodud osa tulemusest (Pilt 31):

	id	Nimi	Režissöör	Pealkiri
1	97	Luis Bu¤uel	97	Bram Stoker's Dracula
2	97	Luis Bu¤uel	97	Gothic
3	227	4AgXpxu3KWgLWw8h1XYh 3KvsOShCKesK6vVU2tceK8	227	The Deep
4	116	bM01iFnt406d rExE8	116	Blue Thunder
5	208	3GAcM9iSxX6l8FM3Wdssfim iwrv0y8mtalAuYhgG95Pmi	208	Shoot to Kill
6	240	LC9X7RMhYVBA4S60LfjleITwoH FgP3rCnzqwZPR3O	240	Twister
7	230	3DzlxvZFXrid KbIdlMRIKxtSgdjJFv	230	The Jungle Book
8	109	Anna Foerster	109	The Rescue
9	192	JHUjMoGl QcRsAhMGmMfVxuLzrhrG	192	A Nightmare on Elm Street 5: The Dream Child
10	119	WMT18ZwDoaxeipurJMuTFenoa 3yjkGvNDCzG9	119	Hamlet

Pilt 31. FULL OUTER JOIN lause kasutamise näide.

Teiste sõnadega, `FULL OUTER JOIN` on kahe tabeli ühend, välja arvatud režissööre, kes sündisid enne `1964-04-05`.

Visuaalne `OUTER JOIN` tulemus on toodud pildil 32.



Pilt 32. FULL OUTER JOIN visuaalne tulemus.

- **`CROSS JOIN`**

  See `JOIN` tüüp tagastab tabeli, kus on potentsiaalselt palju ridu. Ridade arv tulemuses on võrdne ridade arvuga esimeses tabelis korratud ridade arvuga teises tabelis. Iga rida on esimese ja teise tabeli kombinatsioon. Näide on toodud päringus 38.

Päring 38:

```
SELECT Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör CROSS JOIN Film;
```

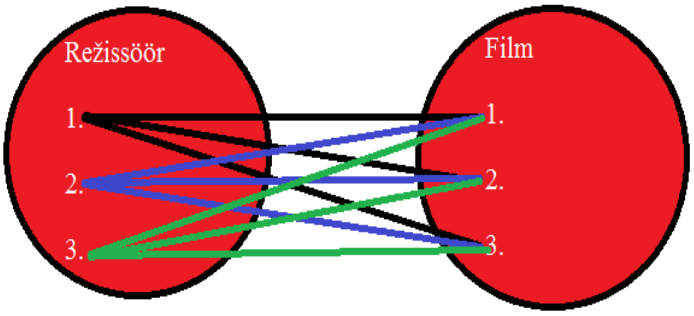
- Execution time: 0.016 seconds

Kuna tulemus on väga mahukas (100 000), allpool on toodud osa tulemusest (Pilt 33):

	id	Nimi	Režissöör	Pealkiri
1	1	Steven Spielberg	176	Fleshburn
2	1	Steven Spielberg	68	First Blood
3	1	Steven Spielberg	63	Rambo: First Blood Part II
4	1	Steven Spielberg	49	Rambo III
5	1	Steven Spielberg	98	Raiders of the Lost Ark
6	1	Steven Spielberg	204	Beethoven
7	1	Steven Spielberg	70	Shinkansen daibakuha
8	1	Steven Spielberg	72	E.T. the Extra-Terrestrial
9	1	Steven Spielberg	87	Terminator 2: Judgment Day
10	1	Steven Spielberg	74	The Terminator

Pilt 33. CROSS JOIN lause kasutamise näide.

Visuaalne `CROSS JOIN` tulemus on toodud pildil 34.



Pilt 34. CROSS JOIN visuaalne tulemus.

3.8.2 *Comma-Separated* JOIN süntaks

Comma-Separated süntaks toetab ainult `INNER JOIN` tüüpi [13]. *Comma-Separated* süntaksi näide on toodud allpool päringus 39.

Päring 39:

```
SELECT f.Pealkiri, o.Kategooria, v.Hinne FROM Film f, Oscar o, Vaatamine v WHERE f.id = o.Film AND f.id = v.Film AND o.Film = v.Film;
```

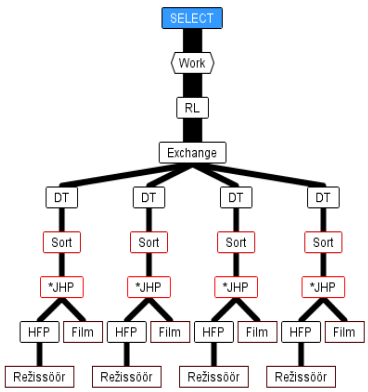
- Execution time: 0.218 seconds

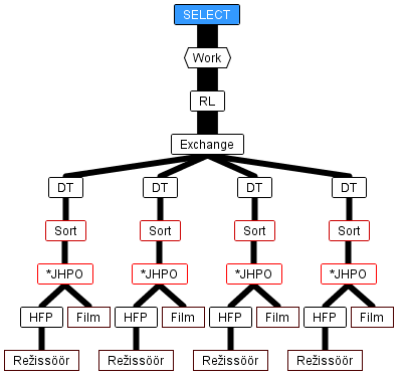
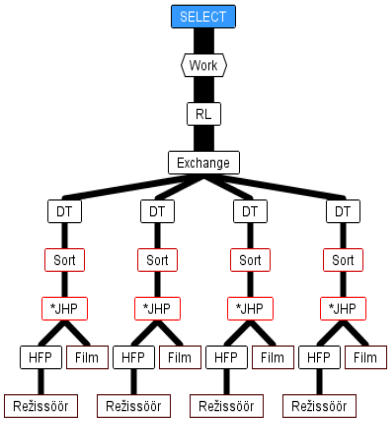
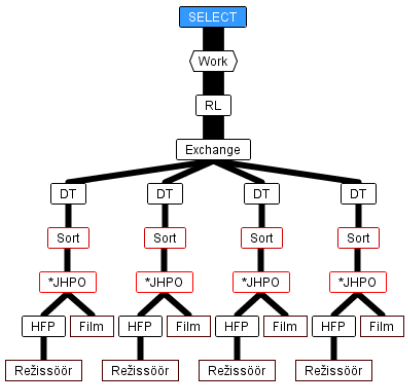
Eespool toodud päringus `WHERE` lause määrab tingimusi, mille alusel kirjed ühendatakse.

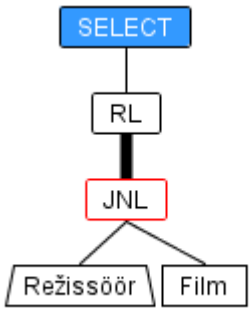
3.8.3 Päringute võrdlus

Parema ülevaade andmiseks on koostatud tabel 6, kus on toodud iga *ANSI* süntaksi `JOIN` tüübi päringud. Iga päring tagastab 8 000 rida.

Tabel 6. JOIN tüüpide ajaline võrdlus.

Rida	`JOIN` tüüp	Aeg (sec)	Puu	Päring
1.	`INNER JOIN`	0.078		SELECT TOP 8000 Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör INNER JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05' ORDER BY Režissöör.id;

2.	LEFT JOIN	0.031		SELECT TOP 8000 Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör LEFT JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05' ORDER BY Režissöör.id;
3.	RIGHT JOIN	0.031		SELECT TOP 8000 Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör RIGHT JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05' ORDER BY Režissöör.id;
4.	FULL OUTER JOIN	0.015		SELECT TOP 8000 Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör FULL OUTER JOIN Film ON Režissöör.id = Film.Režissöör WHERE Režissöör.Sünniaeg > '1964-04-05' ORDER BY Režissöör.id;

5.	CROSS JOIN	0.066		SELECT TOP 8000 Režissöör.id, Režissöör.Eesnimi + ' ' + Režissöör.Perenimi AS Nimi, Režissöör, Pealkiri FROM Režissöör CROSS JOIN Film ORDER BY Režissöör.id;

Analüüsides koostatud tabelit, tuleb välja, et `INNER JOIN` osutus kõige ajamahukamaks, sest sel juhul leitakse kahe tabeli ühisosa. Teine aeglasem ühendamise viis on `CROSS JOIN`, sest selle puhul tehakse kahe tabeli otsekorrutis. Mida mahukam on tabel, seda rohkem aega selleks kulub.

Kõige efektiivsem `JOIN` tüüp on `FULL OUTER JOIN`. Selle põhjuseks on see, et `FULL OUTER JOIN` ei teosta kontrolli, vaid leiab tabelite ühendit. Antud juhul `FULL OUTER JOIN` töötab 0.078 / 0.015 = 5.2 korda kiiremini kui `INNER JOIN`.

Võrreldes omavahel *ANSI* ja *Comma-Separated* `JOIN` lauseid, siis viimane on tunduvalt aeglasem. Seega aja kulu vähendamiseks tuleb võimalusel kasutada *ANSI* `JOIN` päringuid.

3.9 Trigerite muutmine

Trigerid (*ingl.k.* trigger) on salvestatud PL/SQL plokid, mis on seotud tabeliga, skeemiga või andmebaasiga ja käivituvad vastava sündmuse toimumisel [15]. Trigeri loomiseks on kasutajal vaja `CREATE TRIGGER` õigust [15]. Kui triger on loodud, siis andmebaas kompileerib selle automaatselt ning paneb trigeri tööle [15]. Loodud trigeri sisu muutmiseks on vaja kasutada `ALTER TRIGGER` lauset.

Andmebaasis „Film“ on loodud triger nimega „Uuendus“, mis reageerib igale lisamisele või andmete muutumisele tabelis „Film“. Trigeri „Uuendus“ SQL lause on toodud all.

Päring 40:

```
CREATE TRIGGER Uuendus ON Film
FOR INSERT, UPDATE AS
PRINT 'Sa oled lisanud / muutunud andmed tabelis „Film"';
```

- Execution time: 0.19 seconds

Nüüd uute andmete lisamisel tabelisse „Film", loodud triger „Uuendus" käivitub. Järgmises näites lisatakse tabelisse „Film" üks uus rida:

Päring 41:

```
INSERT INTO Film (Pealkiri, Aasta, Riik, Kestus, Keel, Žanr,
Režissöör, Eelarve)
VALUES ('Captain America: Civil War', 2016, 'USA', 120, 'inglise',
'fantaasia', 5, 50000);
```

- Execution time: 0.016 seconds

Lause täitmiseks kulus 41 sekundit. Nüüd triger „Uuendus" oli kustutatud:

Päring 42:

```
DROP TRIGGER Uuendus;
```

Edasi lisati veel üks rida tabelisse „Film":

Päring 43:

```
INSERT INTO Film (Pealkiri, Aasta, Riik, Kestus, Keel, Žanr,
Režissöör, Eelarve) VALUES ('Jungle Book: Maugli', 2016, 'USA', 110,
'inglise', 'fantaasia', 15, 67000);
```

- Execution time: 0.015 seconds

Võrreldes näidete 41 ja 43 täitmiseaega, siis võib teha järelduse, et trigeri töö aeglustab päringu töötlemist. Alati tuleb hoolikalt läbi mõelda, kas triger on kindlasti vajalik või mitte. Kui kasutada liiga palju trigereid, siis andmebaasi produktiivsus langeb. Sellisel juhul on vaja kas muuta või üldse blokeerida trigeri kasutamist. Trigerite blokeerimine terves andmebaasis toimub järgmise lause abil:

```
SET TEMPORARY OPTION FIRE_TRIGGERS = OFF;
```

Kui on ikka vaja trigereid kasutada, siis peab teadma järgmisi reegleid [16]:

- Kasutada trigereid ainult selleks, et kinnitada, kas mingi operatsioon on edukalt lõpetaud oma tööd.
- Mitte luua trigereid, mis kopeerivad juba sisseehitatud funktsionaalsust. Näiteks, ei tasu luua trigerit andmete terviklikkuse kontrollimiseks. Seda on võimalik teha, kasutades deklaratiivset terviklikkuse päringut.
- Piirata trigerite suurust (60 rida või vähem). Kui loogika nõuab rohkem ruumi, siis on vaja panna osa koodist protseduuri ja kutsuda esile see protseduur trigeris.
- Mitte kasutada rekursiivset trigerit. Näiteks, `AFTER UPDATE` trigeri loomine (mis ise kutsub `UPDATE` avaldust) paneb trigeri rekursiivselt käivituma nii kaua, kui mälu jätkub.

3.10 Kitsenduste muutmine

Kitsendused (*ingl.k.* constraints) [18] on andmebaasi skeemi definitsiooni osa. Need on piirangud, mis on kehtestatud tabeli veergudele. Neid kasutatakse selleks, et piirata andmete väärtusi, mida saab panna tabelisse. See aitab kindlustada andmete täpsust ja usaldusväärsust andmebaasis. Kitsendused on tavaliselt seotud tabelitega ning loodud `ADD CONSTRAINT` lause abil. Nad määratlevad teatud omadusi, millele peavad andmed andmebaasis vastama. Kitsendusi saab rakendada ühele veerule, tervele tabelile, mitmele tabelile või tervele skeemile. On olemas mitu kitsenduse tüüpi:

- `Not null` – ükski väärtus veerus ei saa olla null;
- `Unique` – väärtused iga tabeli atribuudis peavad olema unikaalsed (ei saa korduda);
- `Primary key` – väärtused iga tabeli atribuudis peavad olema unikaalsed (ei saa korduda) ning ei saa olla null; tavaliselt igal tabelil andmebaasis peab olema primaarvõti, mida kasutatakse individuaalsete kirjete identifitseerimiseks.;
- `Foreign key` – väärtusi antud atribuudis tuleb viidata olemasolevale kirjele teises tabelis (läbi selle primaarvõtme või mõne muu unikaalsuse piirangu);
- `Check` – avaldus määrab, mida tuleb kontrollida, et kindlustada, kas piirang on täidetud või mitte;

- `Default` – määrab väärtust, mis on pandud tabelisse vaikimisi.

SQL kitsendust kasutatakse reeglite täpsustamiseks tabeli andmete jaoks. Kitsendus saab olla määratud tabeli loomisel või võib-olla hiljem lisatud. Kitsendusi ei saa blokeerida, neid saab ainult kustutada järgmise SQL lause abil:

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name
```

On olemas ka teine variant kitsenduse eemaldamiseks:

`ALTER TABLE table_name DROP FOREIGN KEY (column1,…);` , kus
column1 on veeru nimi, kus asub välisvõti.

Kui on vaja lisada kitsendust, tuleb kasutada lauset:

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name
```

Kitsenduse loomise süntaks on toodud allpool:

```
[ CONSTRAINT constraint-name ] { UNIQUE ( column-name [ , … ] ) |
PRIMARY KEY ( column-name [ , … ] ) | foreign-key-constraint | CHECK
( condition ) }
```

Selleks, et vaadata, mis kitsendused on juba tabelis olemas, tuleb kasutada sisse ehitatud funktsiooni `sp_helpconstraint` järgmise süntaksiga:

```
sp_helpconstraint [objname] , detail.
```

3.11 Tabeli denormaliseerimine

Date (2007) esitab formaalse denormaliseerimise definitsiooni:

„Olgu R_1, R_2,..., R_n relatsioonid. Sellisel juhul tähendab nende denormaliseerimine järgmist: relatsioonid asendatakse nende ühendiga R nii, et iga võimaliku R_1, R_2, ..., R_n väärtuse r_1, r_2, ..., r_n korral kehtib olukord, et tehes projektsiooni R väärtuse r põhjal üle R_i atribuutide saadakse tulemuseks r_i.“ [19].

Praktiliselt tähendab see tabelite normaalkuju muudatust, lähtudes kasutamise vajadustest. Selle protsessi vastand on normaliseerimine (ehk normaalkujule viimine), mille tulemusena on võimalik vältida andmete vastuolu ning lisamis- ja kustutamisanomaaliaid. Tabeli normaalkuju on tabeli korraldamise reeglid, mis aitavad tabelites olevate andmetega manageerida. Erki Eessaar oma õppematerjalis andmebaaside projekteerimisest kirjeldas denormaliseerimist järgmiselt:

„Denormaliseerimine on protsess, mille käigus vähendatakse ühe või mitme andmebaasis oleva baastabeli (edaspidi tabeli) normaliseerimise astet. Tulemusel kas dubleeritakse veerge või ühendatakse tabeleid, et andmete otsimisel oleks vaja vähem läbi viia tabelite ühendamise operatsioone.“ [18].

Kui denormaliseerimise eesmärgiks on parandada päringute täitmiseaega, siis normaalkuju aitab hallata andmete terviklikkust. Tuleb pöörata tähelepanu, et selle protsessi käigus võib tekkida andmete liiasus, mille kaotamiseks on vaja tabelit hoopis jagada väiksemateks tabeliteks [18].

3.12 Andmete restruktureerimine

Päringu restruktureerimine ei ole ainuke viis tõsta andmebaasi efektiivsust. Andmete ümber paigutamisega on võimalik tõsta töökiirust. Selleks tuleb:

1. Tutvuda saadud väärtustega ning vältida püsiva tabeli loomist, kasutades `GROUP BY` käsku. `GROUP BY` kasutatakse päringu tulemuse rühmitamiseks järgmise süntaksi abil:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;, kus
```

`column_name` – kasutatava veeru nimi;

`aggregate_function(column_name)` – kasutatava funktsiooni nimi;

`table_name` – kasutatava tabeli nimi;

`column_name operator value` – võrdlemis tingimus;

`GROUP BY` eeldab tabeli täisskaneerimist, mis võtab aega ja vähendab andmebaasi jõudlust. Seda operaatorit on hea kasutada siis, kui ühes päringus on mitu agregeerivat funktsiooni. Näiteks on koostatud päring 44, mis näitab viit parimat näitlejat, kellel on kõige rohkem Oscareid.

Päring 44:

```
SELECT TOP 5 Eesnimi + ' ' + Perenimi AS Nimi,
COUNT(Oscar.Näitleja) AS OscarNr, COUNT(Roll.Näitleja) AS Rollid
FROM Näitleja JOIN Oscar ON Oscar.Näitleja = Näitleja.ID JOIN Roll
ON Roll.Näitleja = Näitleja.ID GROUP BY Nimi ORDER BY Nimi;
```

- Execution time: 0.009 seconds

Tulemus on toodud pildil 35:

	Nimi	OscarNr	Rollid
1	08yIujkOUOWP0TTI oF6Q2bnfFYsb5kLm	80	80
2	0hXWXAOn6T aSGJitYY	98	98
3	0j6WzOkcS JHj2FvwPQNMiugcmVbHeeS	165	165
4	0KJ4DXLc7bqcCUq8kXsyISknr V9TgX7dkeTrLWncTbz2YI1v9mw	78	78
5	0m w6jJNuYqVzCZOUMWaXfWl8osDL	90	90

Pilt 35. Päringu 44 tulemus.

Kui päringus on palju `WHERE` tingimusi, siis `GROUP BY` operaator uurib ainult neid kirjeid, mis rahuldavad tingimusi.

2. Vajadusel saab kasutada eraldamist [20]. Punktis 3.11 räägiti, et eraldamist tasub vältida. Kui aga kasutada seda mõistlikult saab parandada andmebaasi juhtimissüsteemi jõudlust. Üldine reegel kõlab järgmiselt:

   kui loodud tabel andmebaasis on sagedasti kasutatav, siis tasub kasutada eraldamist. Kui tabelis olevaid andmeid kasutatakse harva, siis tuleb tabelit ühendada (denormaliseerida).

   Eraldamine (ing. k. *Partitioning*) on tabelite jagamine väiksemateks füüsilisteks struktuurideks, sõltuvalt konkreetse veeru väärtusest. See lihtsustab suurte tabelite haldamist. Samuti, jagamine tõstab jõudlust ning aitab täitmise plaani koostamisel ja päringute täitmisel (sellest räägitakse täpsemalt osas 5). Jagatud tabelit saab teha nii andmebaasi arendamisel, kui ka hiljem, pärast andmete lisamist tabelisse. Tehtud tabeli muutumine on aeglane ja keeruline protsess, aga pärast eraldamist on näha kui palju kiiremini saab vajalike andmeid kätte.

3.12.1 Tabeli fragmenteerimine ja reorganiseerimine

Kui tabelit sageli kasutatakse `UPDATE` ja `DELETE` päringutes, siis võib tekkida tabeli fragmenteerimise oht. Mida rohkem tabelit muudeti, seda suurem on fragmenteerimise oht. Miks tekkib fragmenteerimine? Andmete kustutamise päring ei vabasta mälu blokke, kus eemaldatud andmed olid hoitud. Seega need blokid ei lähe kohe kasutusse. Mida rohkem andmeid on kustutatud, seda rohkem jääb kasutut ruumi (*dead space*). Selle tulemuseks on efektiivsuse langus ning lõpus saab mälu otsa, vaatamata sellele, et andmed ei olnud lisatud andmebaasi. Tuleb pöörata tähelepanu, et fragmenteerimine ei teki andmete lisamisel. Üks võimalus tuvastada fragmenteerimist on vaadata, kas tegelik tabeli suurus vastab oodatud suurusele arvestades andmete mahtu tabelis. Teine viis on kasutada süsteemset protseduuri `sa_table_fragmentation`. Näide on toodud päringus 45.

Päring 45:

```
call sa_table_fragmentation('Film');
```

- Execution time: 0.015 seconds

Päringu 45 tulemus on toodud pildil 36:

	TableName	rows	row_segments	segs_per_row
1	Film	10 003	10 024	1,002099370188943

Pilt 36. Päringu 45 tulemus.

Pildil 36 on näha, et segmentide arv ei ole oluliselt suurem kui ridade arv tabelis. Seega võib teha järelduse, et tabelis veel ei tekkinud suurt fragmenteerimist. Lisaks on oluline teada, kuidas saab tabeli jaotamist parandada. Fragmenteerimise vastu aitab tabeli reorganiseerimine, mille käigus vabastatakse kasutu mälu ning ehitatakse tabeli ümber. Reorganiseerimist saab teha `REORGANIZE TABLE` lause abil [21], mille süntaks on toodud allpool:

```
REORGANIZE TABLE [ owner.]table-name
[ { PRIMARY KEY
| FOREIGN KEY foreign-key-name
| INDEX index-name } ], kus
```

`PRIMARY KEY` - tabeli primaarne võti;

`FOREIGN KEY` - välisvõti;

`INDEX` - indeksi nimi.

Reorganiseerimist tuleb kasutada selleks, et jaotada ridu tabelis või tihendada indekseid, mis on muutunud `DELETE` lause kasutamise tõttu hõredaks. See aitab vähendada lehtede koguarvu, mis oli kasutatud, tabelite ja nende indeksite hoidmiseks. Samuti võib see lause vähendada tasemete arvu indeksite puus. Tuleb märkida, et lause ei vähenda andmebaasi failide koguarvu. Allpool on toodud reorganiseerimise kasutamise näited.

Päring 46:

```
REORGANIZE TABLE Film;
```

- Execution time: 7.986 seconds

Käivitatakse veel kord päringu 45 ning võrdleme tulemust, kui palju mõjutas päring 46. Allpool on toodud päringu 45 tulemus pärast päringu 46 täitmist:

	TableName	rows	row_segments	segs_per_row
1	Film	10 003	10 003	1

Pilt 37. Päringu 45 tulemus pärast reorganiseerimist.

Nüüd on ridade arv tabelis võrdne segmentide arvuga. Seega võib kinnitada, et tabelis „Film“ ei ole fragmenteerimist. Tabelit saab reorganiseerida lähtudes kas primaarvõtmest, välisvõtmest või indeksist. Edasi on toodud reorganiseerimise näidispäringud. Tabeli „Oscar“ fragmenteerimine on näidatud päringus 47.

Päring 47:

```
call sa_table_fragmentation('Oscar ');
```

- Execution time: 0.008 seconds

Tulemus:

	TableName	rows	row_segments	segs_per_row
1	Oscar	9 930	11 230	1,13091641490433

Pilt 38. Tabeli „Oscar“ fragmenteerimise sügavus enne reorganiseerimist.

Edasi reorganiseeritakse tabelit primaarvõtme järgi nagu on näidatud päringus 48.

Päring 48:

```
REORGANIZE TABLE Oscar PRIMARY KEY;
```

- Execution time: 0.803 seconds

Vaadetakse uuesti fragmenteerimise sügavust:

	TableName	rows	row_segments	segs_per_row
1	Oscar	9 930	11 230	1,13091641490433

Pilt 39. Tabeli „Oscar“ fragmenteerimise sügavus pärast reorganiseerimist primaarvõtme järgi.

Võrreldes pilte 38 ja 39, võib teha järelduse, et antud reorganiseerimise valik ei aidanud. Siis rakendatakse tavalist reorganiseerimist päringu 49 abil.

Päring 49:

```
REORGANIZE TABLE Oscar;
```

- Execution time: 1.194 seconds

Ning võrreldakse tulemust, mis on toodud pildil 40 pildiga 38.

	TableName	rows	row_segments	segs_per_row
1	Oscar	9 930	9 930	1

Pilt 40. Tabeli „Oscar“ fragmenteerimise sügavus pärast üldist reorganiseerimist.

Seekord reorganiseerimine aitas ning tabelis „Oscar“ ei esine rohkem fragmenteerimist.

3.13 Andmete vaatamisarvu vähendamine

Mõnikord on vaja saada tulemuseks mitut erinevat olemite hulka tabelite komplektist. Tavaliselt tehakse seda läbi mitme skaneerimise, kuigi palju lihtsam on arvutada välja vajalikud hulgad ühe skaneerimisega. Kui on vajadus tabelist saada mitu erinevat informatsiooni, siis pole mõtet koostada mitut erinevat `SELECT` avaldust. Kogemusega ABJS kasutaja oskab koostada SQL päringut nii, et vaadata andmed nii vähe, kui võimalik ehk laadida iga veergu ainult üks kord [12]. See vähendab võrguliiklust ning andmebaasi koormust. Selleks, et seda saavutada on vaja kombineerida otsinguid `CASE` lausega [12]. N-1 skaneerimise kaotamine võib paratamatult parandada produktiivsust. Erinevaid andmete hulki saab kuvada, kasutades `CASE` ja `WHERE` väljendeid, mis filtreerib andmed hulkade jaoks. Iga hulga jaoks saab olla eraldi veerg, kus asuvad vajalikud andmed. Järgmised päringud 50, 51 ja 52 arvutavad filmide arvu, mille eelarve on suurem kui 5000 eurot, 50000 ja 100000 eurot ning väiksem kui 50000 eurot.

Päring 50:

```
SELECT COUNT (*) as SuurEelarvelisedFilmid
FROM Film
WHERE Eelarve > 50000;
```

- Execution time: 0.015 seconds

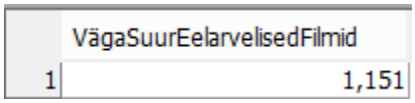
	SuurEelarvelisedFilmid
1	9,857

Pilt 41. Päringu 50 tulemus

Päring 51:

```
SELECT COUNT (*) as VägaSuurEelarvelisedFilmid
FROM Film
WHERE Eelarve BETWEEN 50000 AND 100000;
```

- Execution time: 0.016 seconds



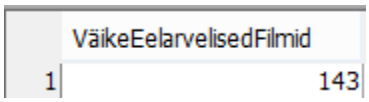
	VägaSuurEelarvelisedFilmid
1	1,151

Pilt 42. Päringu 51 tulemus.

Päring 52:

```
SELECT COUNT (*) as VäikeEelarvelisedFilmid
FROM Film
WHERE Eelarve < 50000;
```

- Execution time: 0.016 seconds



	VäikeEelarvelisedFilmid
1	143

Pilt 43. Päringu 52 tulemus.

Kokku nende kolme päringu täitmine võttis aega 0,015 + 0,016 + 0,016 = 0,047 sekundit. Efektiivsem ja mugavam on kätte saada kõik andmed ühe päringuga. Iga arvu tuleb arvutada eraldi veerus ning arvutamiseks tuleb kasutada operaatorit `CASE`, mis võtab arvesse ainult need read, kus antud tingimus kehtib. Allpool on toodud päring 53, mis arvutab filmide arvu, mille eelarve on suurem kui 5000, 50000 ja 100000 eurot ning väiksem kui 50000 eurot.

Päring 53:

```
SELECT COUNT (CASE WHEN Eelarve > 50000
THEN 1 ELSE null END)
SuurEelarvelisedFilmid,
COUNT (CASE WHEN Eelarve BETWEEN 50000 AND 100000
THEN 1 ELSE null END)
VägaSuurEelarvelisedFilmid,
COUNT (CASE WHEN Eelarve < 50000
THEN 1 ELSE null END)
VäikeEelarvelisedFilmid
FROM Film;
```

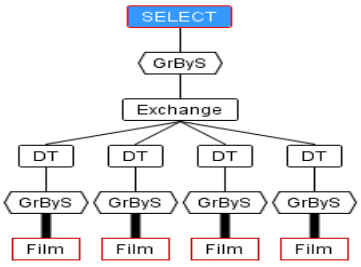
- Execution time: 0.031 seconds

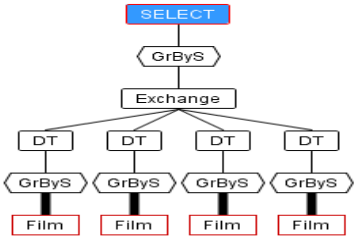
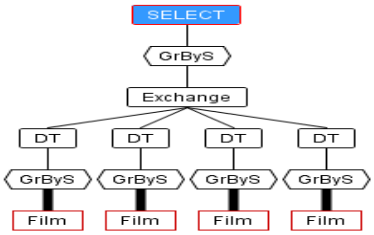
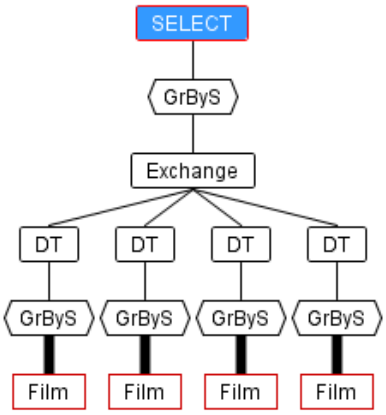
	SuurEelarvelisedFilmid	VägaSuurEelarvelisedFilmid	VäikeEelarvelisedFilmid
1	9,857	1,151	143

Pilt 44. Päringu 53 tulemus.

Tabelis 7 on toodud päringute 50, 51, 52 ja 53 täitmiseajad, tagastatud ridade arv ning puustruktuur, et anda ülevaadet päringute efektiivsusest. Rida 5 näitab päringute 50, 51 ja 52 summaarset informatsiooni.

Tabel 7. Päringute 50, 51, 52 ja 53 efektiivsuse võrdlus.

Rea number	**Päringu number**	**Aeg (sec)**	**Puu struktuur**	**COUNT operaatori tulemus**
1.	50	0.015		9 859

2.	51	0.016		1153
3.	52	0.016		143
5.	50,51,52	0.047		11 155
4.	53	0.031		11 155

Võrreldes päringut 50, 51 ja 52 summaarset aega, mis on 0.047 sekundit, siis saab teha järelduse, et päring 55 on 0.047 / 0.031 = 1.5 korda efektiivsem. Vaadates eelpool toodud päringute puustruktuure, siis päringu 53 puu ei erine päringute 50, 51 ja 52 puudest, aga päringu 53 täitmine on tunduvalt kiirem. Lisaks ühte päringu kirjutame nõuab kasutajalt vähem aega, kui kolme erinevate päringute kirjutamine.

4. Tabelite indekseerimine

Andmebaasist andmete otsimist saab võrrelda raamatukogus raamatu otsimisega. Selleks, et leida huvi pakkuv raamat, tuleb vaadata konkreetses osakonnas ning seejärel otsida raamat ID järgi. Näiteks, keegi ei hakka anatoomia õpikut otsima matemaatika osakonnast, otsitakse ikka meditsiini õpikute osakonnast. Sellise loogikaga saab pöörduda andmete poole andmebaasis. Alati võib vaadata kõik read läbi, aga palju efektiivsem on kasutada indekseid. Nende eesmärk on tõsta andmebaasi jõudlust ja vähendada päringute töötlemise aega. Järgnevas peatükis kirjeldatakse vastavaid võimalusi *SQL Anywhere* andmebaasi juhtimissüsteemis ning indeksite plusse ja miinuseid.

4.1 Indeksite loomine tabelites

Indeks on andmebaasi objekt, mille eesmärk on kiirendada andmebaasis andmete otsimist. Indeksi loomisel määratakse struktuur ja antakse unikaalne nimi, mida koostatakse kokkulepitud reeglite järgi. Indeksi struktuur on andmebaasi juhtimissüsteemides erinev, kuid reeglina võiks see sisaldada andmefaili identifikaatorit, tabeliploki identifikaatorit ja identifikaatorit, mis näitab rea positsiooni plokis [24].

Priit Raspel kirjutab oma õppematerjalis järgmiselt: *„Esimene veerg sisaldab otsingu võtit ja teine sellele võtmele vastava kirje aadressi tabelis. Uue kirje lisamisel tabelisse kirjutatakse rida ka igasse indeksisse, mis on selle tabeliga seotud. Igasse indeksisse rea lisamisel moodustatakse tabelisse lisatud kirje väärtustest indeksi võtme kirjelduse alusel võtme väärtus, loetakse kirje aadress ja kirjutatakse need väärtused indeksisse - võtme väärtus esimesse veergu ja aadress teise veergu.“* [24].

Indekseid saab luua ka siis, kui tabelis pole veel andmeid. Indeksi loomise süntaks on järgmine:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WHERE <filter_predicate> ]
```

[WITH (<relational_index_option> [,...n])]

[;], kus

[UNIQUE] [CLUSTERED | NONCLUSTERED] on indeksi tüüp,

<object> - objekt, millele luuakse indeksit;

INCLUDE - veerud, mis ka seotud selle indeksiga;

WHERE - tingimused, millele objekt peab vastama, et luua indeksit;

WITH - indeksi lisa parameetrid. Näiteks, maksimaalne ridade arv.

Päring 54 näitab indeksi loomist tabelile „Film“ veerule „Režissöör“.

Päring 54:

```
CREATE INDEX NIX_Film_Režissöör ON Film(Režissöör);
```

Edasi rakendatakse loodud indeksit päringule:

Päring 55:

```
SELECT TOP 5 Pealkiri from FILM
WITH (INDEX(NIX_Film_Režissöör))
JOIN Oscar ON Film.id = Oscar.Film;
```

- Execution time: 0.016 seconds

	Pealkiri
1	Fleshburn
2	First Blood
3	Rambo: First Blood Part II
4	Rambo III
5	Shinkansen daibakuha

Pilt 45. Päringu 55 tulemus.

Nüüd täidetakse päringut 56, mis annab sama tulemuse indeksi NIX_Film_Režissöör kasutamata.

Päring 56:

```
SELECT TOP 5 Pealkiri from FILM
JOIN Oscar ON Film.id = Oscar.Film;
```

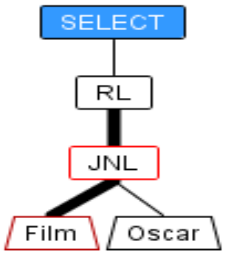
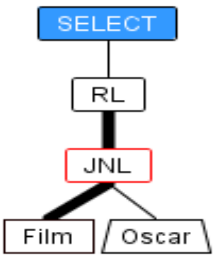
- Execution time: 0.047 seconds

	Pealkiri
1	Fleshburn
2	First Blood
3	Rambo: First Blood Part II
4	Rambo III
5	Shinkansen daibakuha

Pilt 46. Päringu 56 tulemus.

Tabel 8 näitab päringute 55 ja 56 täitmiseaega ning puustruktuurid. Tabeli abil on võimalik analüüsida indeksi kasutamise efektiivsust ning selle mõju päringu puustruktuurile.

Tabel 8. Päringu 55 ja 56 võrdlus.

Rea number	Päringu number	Aeg (sec)	Puu struktuur
1.	55	0.016	
2.	56	0.047	

Võrreldes päringuid 55 ja 56 saab teha järelduse, et indeksi kasutamine kiirendab andmete otsimist. Antud juhul indeksiga päring täideti 0.047 / 0.016 = 2.93 korda kiiremini. Mida mahukam on tulemus, seda rohkem aega võtab selle tagastamine. Seega tasub kasutada indeksit otsingu kiirendamiseks. Antud juhul mõlemate päringute eesmärk on tagastada esimesed viis rida. Võrreldes päringute 55 ja 56 puustruktuure, siis võib teha järelduse, et indeks ei mõjuta puu kuju, aga kiirendab otsingut.

Siiski tasub meeles pidada, et indeksite kasutamine ei ole alati mõistlik. Indeksitel on oma negatiivsed küljed. Täpsemalt indeksite miinustest ja plussidest räägitakse punktis 4.4.

4.2 Indeksite klassifikatsioon

Indekseid võib klassifitseerida mitmel erineval viisil. Üks indeks võib kuuluda korraga mitmesse klassi. Edasi vaadetakse erinevaid indeksite tüüpe.

4.2.1 Liitindeksid ja lihtindeksid

Erinevus liit- ja lihtindeksite vahel on see, et lihtindeks on seotud ühe veeruga. Need on tavalised indeksid, mis omakorda võivad olla unikaalsed, primaarsed jne. Liitindeksid hõlmavad rohkem, kui ühe veeru. Liitindekseid kasutatakse kas päringu täitmise aega vähendamiseks või ridade unikaalsuse kindlustamise eesmärgil [26].

On toodud tabel „Näitleja“. Veerg „Rahvus“ on mittekohustuslik täitmiseks, seega mõned kirjed võivad olla tühjad. Kui luua veerule „Rahvus“ indeks, siis väärtust `NULL` ei indekseerita; andmete otsing indeksi abil ei kiirene. Otsingut saab teostada sama hästi ilma indeksita. Näide on toodud päringus 57.

Päring 57:

```
SELECT Count(*) AS arv FROM Näitleja WHERE Rahvus = 'inglane';
```

- Execution time: 0.047 seconds

Allpool on toodud olukorrad, kus on vaja kasutada liitindeksit [26]:

Päring 58:

```
SELECT * FROM Režissöör WHERE eesnimi='Jaan' AND Rahvus = 'eestlane';
```

- Execution time: 0.062 seconds

Veergude kombinatsioon annab oluliselt parema selektiivsuse kui iga veerg eraldi. Tüüpilised päringud kasutavad päringu tingimuses ühte ja sama veergude komplekti vaadeldavast tabelist. Sellise veergude komplekti koondamisel indeksisse saavutatakse seda, et päringud saavad vastused kätte otse indeksist, ilma tabeli poole pöördumata.

Liitindeksite loomisel tuleb pöörata tähelepanu veergude järjestusele. On toodud tabel „Näitleja" primaarvõtmega ID. Tabelis on veel olemas veerud „Sünniaeg", „Rahvus", „Sugu", „Eesnimi" ja „Perenimi". Sellele tabelile luuakse indeks NIX_Näitleja_ÜldAndmed allpool toodud lausega 59.

Päring 59:

```
CREATE INDEX NIX_Näitleja_ÜldAndmed(Sünniaeg, Eesnimi, Perenimi);
```

Loodud indeksit saab kasutada, näiteks allpool toodud päringutes 60, 61 ja 62.

Päring 60:

```
SELECT * FROM Näitleja WHERE Sünniaeg='1994-05-06';
```

- Execution time: 0.032 seconds

Päring 61:

```
SELECT * FROM Näitleja WHERE Eesnimi='Roger' AND Perenimi='Dickson';
```

- Execution time: 0.032 seconds

Päring 62:

```
SELECT * FROM Näitleja WHERE Sünniaeg='1994-05-06' AND
Eesnimi='Roger' AND Perenimi='Dickson';
```

- Execution time: 0.062 seconds

Tuleb pöörata tähelepanu, et indeksi liigkasutamine vähendab päringu täitmise aega. Tabel 9 näitab päringute 60, 61 ja 62 täitmiseaega, puustruktuuri ning tagastatud ridade arvu. Selle abil on võimalik analüüsida indeksi mõju päringu täitmisele.

Tabel 9. Päringute 60, 61 ja 62 võrdlus.

Rea number	Päringu number	Aeg (sec)	Puu struktuur	Tagastatud ridade arv
1.	60	0.032		1
2.	61	0.032		0
3.	62	0.062		0

Sarnaselt tabeliga 8, indeksi loomine ei mõjuta puustruktuuri. Kõikidel eelpool mainitud päringutes puustruktuur on sama.

Kokkuvõtteks võib öelda, et enne indeksi kasutamist, tuleb hoolikalt läbi mõelda päringu ja indeksi eesmärgid, sest indeksite liigkasutamine oluliselt aeglustab päringute töötlemise aega. Kuid indeksite oskuslik kasutamine tõstab päringu efektiivsust.
Kui kasutada päringus veergu, mis ei kuulu NIX_Näitleja_ÜldAndmed indeksisse, siis sellel juhul liitindeksit ei kasutata. Näide on toodud allpool päringus 63.

Päring 63:

```
SELECT * FROM Näitleja WHERE Rahvus= 'ameeriklane';
```

- Execution time: 0.031 seconds

Tabelile võib vajadusel luua ka mitu liitindeksit [26].

4.2.2 Unikaalsed ja mitteunikaalsed indeksid

Indekseid jaotatakse indekseeritud veerus olevate väärtuste unikaalsuse järgi. Sinna kuuluvad:

- Unikaalsed indeksid, mida omistatakse primaarvõtme või unikaalsuse kitsendusega seotud veergudele;
- Mitteunikaalsed indeksid (primaarvõtme või unikaalsuse kitsenduse poolt sidumata veergudele).

Andmebaasi juhtimissüsteem võib kasutada indekseid, et kiirendada kitsenduste kontrolli [26]. Kui tabeli loomisel või muutmisel defineerida tabelile primaarvõti (`PRIMARY KEY` kitsendus) või lisada mõnele veerule unikaalsuse nõue (`UNIQUE` kitsendus), siis peaaegu kõik andmebaasi juhtimissüsteemid loovad nendele veergudele automaatselt unikaalse indeksi [26]. Tänu indeksi olemasolule saab andmebaasi juhtimissüsteem indekseeritud veergu väärtuse lisamisel või muutmisel kontrollida, kas unikaalsus on säilinud või mitte [26].

4.2.3 Primaarsed ja sekundaarsed indeksid

Primaarsed ja sekundaarsed indeksid klassifitseeritakse veeru tüübi järgi, millele on indeks loodud. Primaarne indeks luuakse primaarvõtmes osalevate veergudele. Tabelil saab olla maksimaalselt ainult üks primaarvõti ning primaarne indeks [26]. Sekundaarne indeks luuakse primaarvõtmes mitteosalevate veergudele. Tabelis võib olla mitu sekundaarset indeksit [26].

4.2.4 Mitte rühmitatud ja rühmitatud indeksid

Indeks määratleb andmete hoidmise loogikat. Ühes tabelis saab olla rohkem kui üks mitte rühmitatud indeks. Mitte rühmitatud (*non-clustered*) indeksite puustruktuuris olevad indeksite võtmed on sorteeritud ning iga taseme leht omab salvestamise viite (näiteks, rea numbrit) [27]. Rühmitamine paneb andmete blokid konkreetsesse järjekorda, mis sobitub indeksit järjekorraga. Rühmitamise tulemuseks on saadud rida, kus kogu informatsiooni hoitakse ettemääratud järjekorras. Seega ühes tabelis saab olla ainult üks rühmitatud indeks. Rühmitatud indeksid saavad tunduvalt tõsta andmebaasi jõudlust kolmel juhtudel:

1. kui andmeid päritakse samas järjekorras, milles andmed on seotud indeksiga;
2. kui andmeid päritakse vastupidises järjekorras, milles andmed on seotud indeksiga;
3. kui piiritletakse tagastatud ridade arvu.

Kuna andmed segmendil on järjestatud, siis iga järgmine rida järjekorras on kättesaadav kohe enne või pärast eelmist rida. Niisugune järjestamine vähendab andmete vaatamisarvu päringu töötlemisel. Seega võib ütelda, et rühmitatud indeksi põhieesmärk on järjestada andmeid vastavalt indeksi plokkide järjekorrale [27]. Rühmitatud indeksi peamine eelis rühmitamata indeksi ees seisneb just füüsiliste ridade järjestamises vastavalt andmete plokkidele, mis viitavad vajalikele andmetele. Jõudluse tõstmiseks tuleb kasutada rühmitatud indekseid, mida hoitakse samal segmendil, kus indeksitega seotud andmed. Kui kasutaja loob rühmitatud indeksi, siis liigutakse terve tabel konkreetsele **segmendile** ning tekitatakse indeksite puu sellele segmendile. Efektiivsuse mõttes tasub panna mitterühmitatud indekseid eraldi segmendile.

4.2.5 Indeksite valimine

Kui indeksit ei ole kasutatud, siis päringu täitmisel vaadatakse andmeid üle kogu tabeli [24]. Terve tabeli üle vaatamist nimetatakse täisskaneerimiseks (ingl. k. *full table scan*). See tähendab, et kui tabelist otsitakse sobivat rida, siis vaadatakse ükshaaval kõik selle tabeli read. Tabelis 10 on toodud omadused, mida saab kasutada iga indeksi tüübi puhul.

Tabel 10. Dublikaatide kontroll iga indeksi tüübi puhul.

Indeksi tüüp	Omadus
Rühmitatud	ignore_dup_row, allow_dup_row
Unikaalne rühmitatud	ignore_dup_key
Mitte-rühmitatud	Pole
Unikaalne mitte-rühmitatud	Pole

Selleks, et vaadata millised indeksid on loodud, on olemas protseduuri `a_table_stat()`. Selle kasutamise näide on toodud päringus 64.

Päring 64:

```
SELECT sa_table_stats.table_name AS table_name,
sa_table_stats.count AS rows,
sa_table_stats.table_page_count AS table_pages,
sa_table_stats.ext_page_count AS ext_pages
FROM sa_table_stats()
WHERE sa_table_stats.creator = 'DBA'
ORDER BY table_name;
```

- Execution time: 0.016 seconds

Pildil 47 on toodud päringu 64 tulemus:

	table_name	rows	table_pages	ext_pages
1	Film	10,003	228	1
2	Näitleja	10,000	144	31
3	Oscar	9,930	88	12
4	Režissöör	10,001	145	42

Pilt 47. Päringu 64 tulemus.

Järgmise SQL päringu 65 abil saab vaadata, millised indeksid on igas tabelis.

Päring 65:

```
execute sa_index_density('Film');
```

- Execution time: 0.05 seconds

Päringu 65 tulemus on toodud pildil 48:

	TableName	TableId	IndexName	IndexId	IndexType	LeafPages	Density	Skew
1	Film	724	Film	0	PKEY	4	0.54681396484375	1.525227152556233
2	Film	724	fk_Lavastab	1	FKEY	18	0.960910373263889	1.046322778506443
3	Film	724	indexFilm	2	NUI	95	0.64968904194079	1.301582780578935
4	Film	724	IX_Film_Režissöör	3	NUI	18	0.960910373263889	1.046322778506443

Pilt 48. Päringu 65 tulemus.

Päringu 66 abil on võimalik kontrollida, kui palju on indeks killustatud.

Päring 66:

```
execute sa_index_levels('Film');
```

- Execution time: 0.013 seconds

Päringu 66 tulemus on toodud pildil 49:

	TableName	TableId	IndexName	IndexId	IndexType	Levels
1	Film	724	Film	0	PKEY	2
2	Film	724	fk_Lavastab	1	FKEY	2
3	Film	724	NIX_Film_Režissöör	3	NUI	2

Pilt 49. Päringu 66 tulemus.

4.3 Indeksite eemaldamine

Enne indeksi loomist tuleb veenduda, et indeksi lisamine aitab parandada süsteemi töökiirust. Lisaks tuleb koostada nimekiri veergudest, mida võib indekseerida. Seejärel on vaja eemaldada indekseid sellistest veergudest, mille **indekseerimine** aeglustab süsteemi tööd. Enamik andmebaasidest (ka *SQL Anywhere*) ei kontrolli indeksi loomisel, kas indekseeritava veeru jaoks on juba indeks loodud või mitte. Niimoodi võivad tekkida liigsed indeksid. Selline olukord mõjutab negatiivselt süsteemi tööd, sest dubleeritud indeksid [26]:

- kasutavad salvestusruumi ebaefektiivselt;
- aeglustavad SQL päringute täitmist ;

- ajavad päringu optimeerijat segadusse, kuna kompilaatoril on võimalus valida mitme samaväärsete indeksite vahel, mis kasutavad sama täitmisplaani.

Indeksid saavad eksisteerida ainult koos tabeliga, tabeli kustutamisel kaovad andmebaasist kõik selle tabeli indeksid. Kui tabelist kustutatakse kirje, siis kustutatakse kõik selle kirjega seotud read kõikidest selle tabeliga seotud indeksitest [24]. Kui kirjes muudetakse veergude väärtusi, mida on kasutatud mõne indeksi võtme moodustamiseks, siis nende kirjetega seotud indeksites kustutatakse vana indeksi kirje ja kirjutatakse selle asemele uue indeksi kirje [24]. On olemas indekseid, kus kirjeid grupeeritakse ja indeksisse kirjutatakse üks rida iga kirjete grupi kohta. Sellistes indeksites ei viita aadress-väli konkreetsele kirjele vaid hoopis kirjete grupi esimesele kirjele. Indeksi eemaldamiseks kasutatakse `DROP` lauset, mille süntaks on järgmine:

```
DROP INDEX table_name.index_name
[, table_name.index_name] ...
```

Kui kasutada sellist lauset, siis vabaneb kohe mälu, mida saab hiljem kasutada teiste objektide loomiseks, mille tõttu väheneb tabeli fragmenteerimise risk (vt. Punkt 3.12.1). Tuleb teada, et sellise lausega ei saa kustutada süsteemi indekseid ning samuti ei saa `DROP INDEX` lausega kustutada indekseid, mis toetavad unikaalsuse kitsendusi. Niisuguste indeksite eemaldamiseks tuleb alguses kustutada kitsendus, kasutades `ALTER TABLE` lauset ning pärast saab kustutada indeks. Lisaks ei saa eemaldada indekseid, mis on hetkel tegevuses. Selleks, et teada saada kas on indeks kinni või ei ole, on olemas `SP_CURSORINFO` protseduur. Indeksite eemaldamiseks peavad kasutajal olema vastavad õigused. Indekseid saab kustutada ainult tabeli omanik.

4.4 Indeksite plussid ja miinused

Indeksite kasutamine ei ole alati ideaalne viis andmebaasi juhtimissüsteemi jõudluse parandamiseks. Allpool on toodud indekseerimise positiivsed ja negatiivsed küljed [28].

Miinused:

- Indeksitega toimub andmete kirjutamine tabelisse aeglasemalt, kuna andmebaas peab eelkõige läbi vaatama kõik selle tabeliga seotud indekseid;
- `INSERT`, `UPDATE` ja `DELETE` laused töötavad aeglasemalt;

- Iga indeks vajab kohta muutmälus (*Random Access Memory*). Indeks, mida ei saa hoida muutmälus on kasutu.
- Kuigi indeksid annavad võimaluse kiiremini otsida vajalikku informatsiooni, siiski suurendavad nad päringu kompileerimise aega.

Plussid:

- Indeksitega töötavad päringud kiiremini. Indeksitega luuakse teist väiksema sügavusega puud, mis kasutab kiiremat otsingut.

Kokkuvõttes saab öelda, et parema tulemuse saavutamiseks tuleb uurida juba andmebaasi projekteerimise ajal andmebaasi kasutamise eesmärke ning lähtuvalt sellest lisada indekseid.

4.5 Praktilised näited

4.5.1 Indeksite kasutamine

Antud näites on kasutusele võetud tabel „Režissöör“, mis sisaldab andmeid režissööride kohta. Tabeli loomise päringut saab vaadata Lisas 2. Päringu 67 abil leitakse, kui palju režissööre on sündinud enne 1990 aastat.

Päring 67:

```
SELECT COUNT(*) FROM Režissöör WHERE Sünniaeg > YEAR(1955) AND
Sünniaeg < YEAR(1990);
```

- Execution time: 0.01 seconds

Pärast veeru „Sünniaeg“ indekseerimist päringu 68 abil jooksutati päring 67 veel kord.

Päring 68:

```
CREATE INDEX SünniaegInd ON Režissöör (Sünniaeg);
```

Päringu 67 täitmise aeg pärast indekseerimist on 0,0009 sekundit. Selle järgi võib teha järelduse, et päringu täitmiseaeg on 0.01/0.009 = 1.(1) korda väiksem.

4.5.2 Andmete järjestamine

Päringute 69 ja 71 eesmärk on leida esimesed sada režissööri ning sorteerida tulemust täisnime järgi. Päring 69 ei kasuta otsimiseks indeksit.

Päring 69:

```
SELECT TOP 100 Eesnimi + ' ' + Perenimi As Nimi FROM Režissöör ORDER
BY Nimi;
```

- Execution time: 0.015 seconds

Kuna päringu 69 tulemus on mahukas (100 rida), pildil 50 on toodud päringu 69 osa tulemusest.

	Nimi
1	0Yl mVq7PBvfDf1t6nJx
2	0YG0pF2Ka8QQEazCsBbZStDuiK AwnGnV
3	0YB8ZSP3SjR9sVC1E zdPCgqNnljBPPXsYsVuzdT
4	0Y3W2RnJDs5arj7cu O8tLeFls3
5	0Y1Z6 zHemsdc0aduFydkREjNzd
6	0Opg2KhOADZgli RfP91BmxzQBoiAKKNFse
7	0oKzb cskJq9lPcRL4sktcWAuRn7xd
8	0OdS91ckgsMPIrm EFTofpK8Z7X
9	0odEUq8cKMEto2rXjkhhPTb9J1v 5F5oNCxktnICBVpO3hg5Paimw
10	0o4U IyFqOtko9gkvXikG8l9eAUdAxdF
11	0nVbPD6gVElRQtffucN4 ZwPC6Kciq1
12	0NDz0nGKHrd0Gm KLAkhmWQQtXZy

Pilt 50. Päringu 69 tulemus.

Lause 70 abil oli loodud indeks nimega UN_Režissöör_Perenimi, mida kasutatakse päringus 71.

Päring 70:

```
CREATE INDEX UN_Režissöör_Perenimi ON Režissöör(Perenimi);
```

Päring 71 tagastab sama tulemuse, mis päring 69. Vahe päringute 69 ja 71 vahel seisneb selles, et päring 71 kasutab päringus 70 loodud indeksit UN_Režissöör_Perenimi.

Päring 71:

```
SELECT TOP 100 Eesnimi + ' ' + Perenimi AS Nimi FROM Režissöör
WITH(INDEX(UN_Režissöör_PereNimi)) ORDER BY Nimi;
```

- Execution time: 0.001 seconds

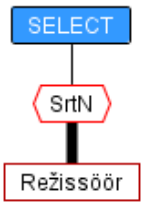
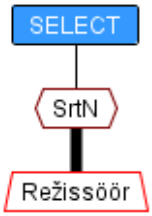
Kuna päringu 71 tulemus on mahukas (100 rida), siis pildil 51 on toodud päringu 71 osa tulemusest.

	Nimi
1	0Yl mVq7PBvfDf1t6nJx
2	0YG0pF2Ka8QQEazCsBbZStDuiK AwnGnV
3	0YB8ZSP3SjR9sVC1E zdPCgqNnljBPPXsYsVuzdT
4	0Y3W2RnJDs5arj7cu O8tLeFls3
5	0Y1Z6 zHemsdc0aduFydkREjNzd
6	0Opg2KhOADZgli RfP91BmxzQBoiAKKNFse
7	0oKzb cskJq9lPcRL4sktcWAuRn7xd
8	0OdS91ckgsMPIrm EFTofpK8Z7X
9	0odEUq8cKMEto2rXjkhhPTb9J1v 5F5oNCxktnICBVpO3hg5Paimw
10	0o4U IyFqOtko9gkvXikG8l9eAUdAxdF
11	0nVbPD6gVElRQtffucN4 ZwPC6Kciq1
12	0NDz0nGKHrd0Gm KLAkhmWQQtXZy

Pilt 51. Päringu 71 tulemus.

Tabel 11 näitab päringute 69 ja 71 täitmiseaega ning puustruktuure.

Tabel 11. Päringute 69 ja 71 võrdlus.

Rea number	Pärigu number	Aeg (sec)	Puu struktuur
1.	69	0.015	
2.	71	0.001	

Võrreldes eelpool toodud andmeid tabelis 11, võib teha järelduse, et järjestatud päringu töötlemine toimib kiiremini. Mõlemad päringud 69 ja 71 tagastasid 100 rida, kuid päringu 71 täitmiseaeg oli 0.015 / 0.001 = 15 korda väiksem, kui päringu 69. Põhjus on selles, et enne otsimist, *SQL Anywhere* järjestab mälus ridu, mille jaoks kulub aega. Kui mälu ei ole piisavalt suur, siis *SQL Anywhere* kirjutab üle kõik read ajutisele segmendile, järjestab ja siis töötleb päringut.

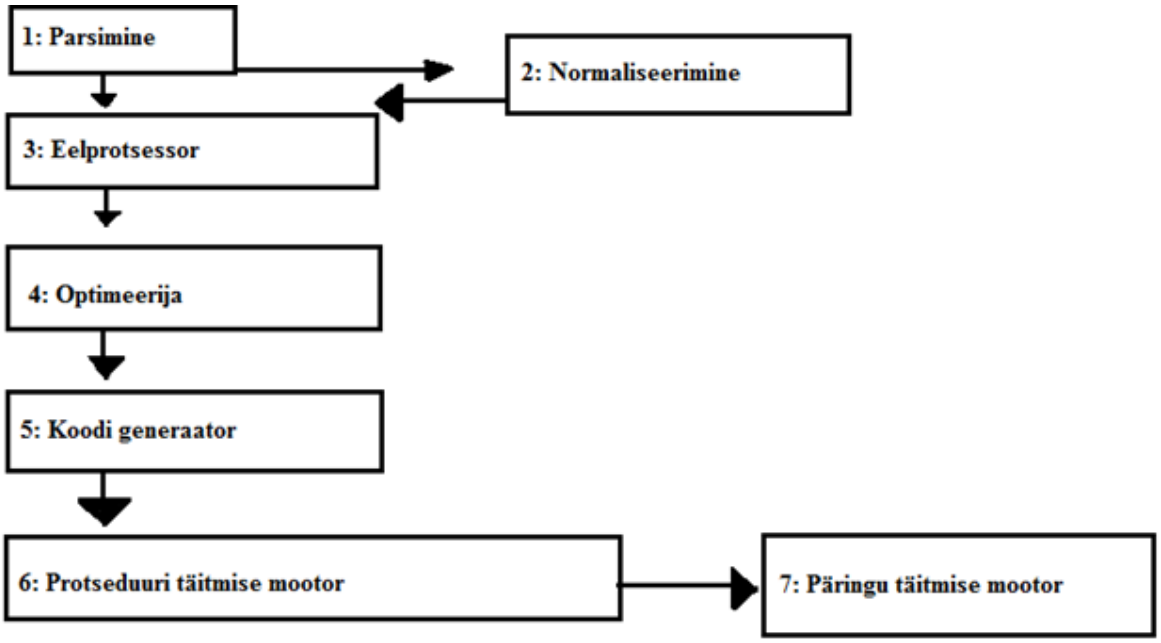
Teine oluline järeldus on see, et indeksi loomine ei mõjuta puustruktuuri, mis oli juba näidatud tabelite 8 ja 9 näidete puhul. Päringute 69 ja 71 puud on samasugused.

5. *SQL Anywhere* päringu planeerija

SQL Anywhere optimeerija vaatab iga kasutaja poolt sisestatud lause üle ning koostab optimeerimise plaani, mis teeb lause töötlemist efektiivsemaks. Graafiline päringu planeerija järgi võib otsustada, kas tabelis on vaja indeksit eemaldada või mitte. Antud peatükis kirjeldatakse päringu planeerija põhiomadusi ja tuuakse näiteid.

5.1 Päringu töötlemine

SQL Anywhere sisene optimeerija koostab plaani iga päringu töötlemiseks. Selleks kasutab optimeerija kasutaja poolt sisestatud päringut. Tulemuse genereerimiseks kasutab päringu planeerija tabelite, indeksite ja veergude statistikat ning erinevaid konfiguratsioone, mis saavad olla muudetud andmebaasi kasutaja poolt. Pildil 52 on toodud etapid, mille järgi koostetakse optimeerimise plaani [8].



Pilt 52. Päringu täitmise plaan [8].

1. *Parser* e. süntaksianalüsaator teisendab päringut puuks.
2. Puud normaliseeritakse. Puuks teisendamine hõlmab tabelite veergude ja nimede määratlemist, puu teisendamist konjunktiivseks normaalkujuks ning andmetüüpide lahendamist.
3. Eeltöötluse etapil muudetakse päringu puu SQL avaldisteks, et optimeerida puustruktuuri.
4. Optimeerija analüüsib kõike võimalikke operatsioonide kombinatsioone (`JOIN` lauseid, paralleelseid protsesse jne) päringute töötlemiseks ning valib kõige sobivama variandi selle hulgast.
5. Koodi generaator teisendab päringu plaani sobivaks formaadiks.
6. Protseduuri mootor täidab selliseid avaldisi nagu tabelite loomine, protseduuride väljakutsumine ning deklareerib otseselt kursori. Selliste DML-i avaldiste jaoks nagu `SELECT, INSERT, DELETE` ja `UPDATE`, loob protseduuri mootor täitmise keskkonna kõikide päringu plaanidele ning käivitab päringu täitmise mootori.
7. Päringute täitmismootor täidab sammud, järgides koodi generaatori plaani.

5.2 Päringu planeerija *Plan Viewer*

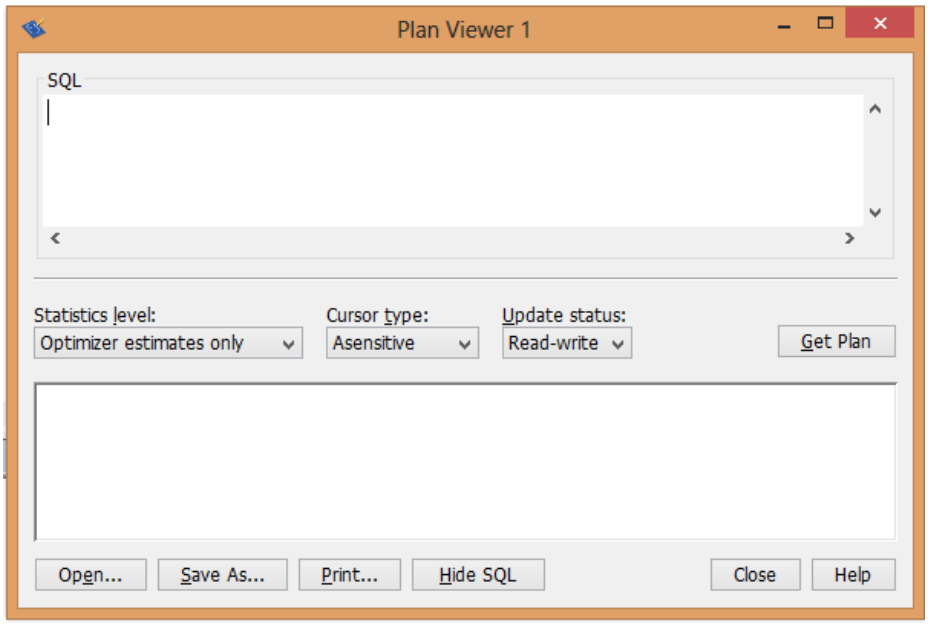
Optimeerija aitab tõsta sisestatud päringu efektiivust, koostades plaani, mille järgi seda päringut töödeldakse. Kasutaja saab teha plaani veelgi efektiivsemaks, vaadates üle järgmised punktid [29]:

- plaan peab valima tabeli jaoks parima filtri;
- `JOIN` järjekord peab tagastama igal sammul minimaalse ridade arvu;
- vaated peavad olema efektiivselt kasutatud;
- igale tabelile peab olema efektiivne juurdepääs.

Päringu plaani muutmisel tuleb arvestada sellega kui palju predikaate on päringus ja ridu tabelis. Eelkõige tasub jälgida kahtlast aktiivsust. Näiteks, mõnedel juhtudel tasub vältida suure ridade arvuga terve tabeli skaneerimist. Kui plaanis on lisatud terve tabeli skaneerimine, tasub kontrollida enne plaani muutmist, miks koostatud plaanis ei kasutata indeksit. Terve tabeli skaneerimine ei tähenda alati ebaefektiivsust.

5.3 Praktilised näited

Antud osas luuakse ja analüüsitakse päringu töötlemise plaani *Plan Viewer*-i abil. Selleks, et avada *Plan Viewer*, tuleb avada *Interactive SQL* aken. Selleks tuleb peaaknas valida *'Tools' -> 'SQL Anywhere 17' -> 'Open Interactive SQL'* ja sisestada andmebaasi kasutaja andmed. Avatud aknas tuleb avada *Plan Viewer* (vt. Pilt 53).



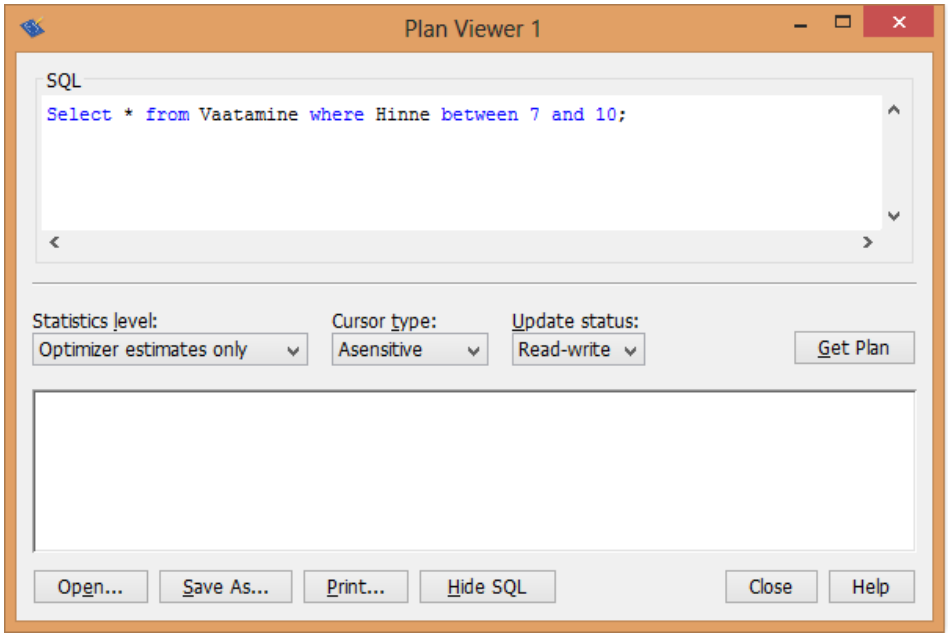
Pilt 53. *Plan Viewer*-i aken.

Edasi tuleb sisestada (või kleepida) päring, mille täitmise plaani on vaja saada. Esimeseks näiteks oli sisestatud üherealine päring 72.

Päring 72:

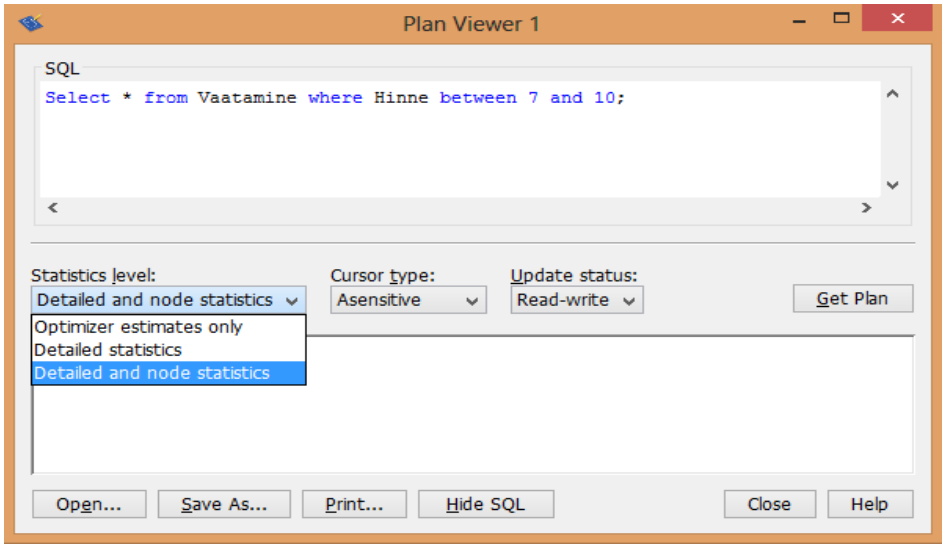
```
SELECT * FROM Vaatamine WHERE Hinne BETWEEN 7 and 10;
```

- Execution time: 0.04 seconds



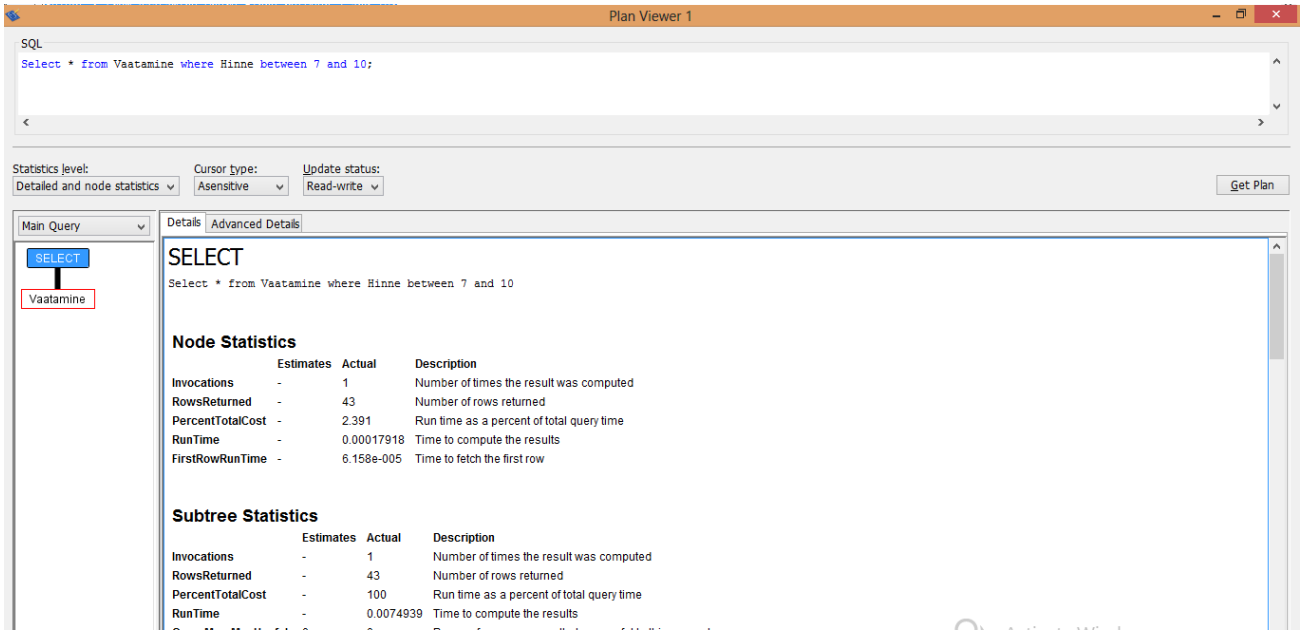
Pilt 54. SQL päringu sisestamise lahter *Plan Viewer*'is.

Tuleb veenduda, et *'Statistics level'* on seadistatud *'Detailed and node statistics'* peale.



Pilt 55. Statistika taseme seadistamine *Plan Viewer*-i liideses.

Plaani saamiseks tuleb vajutada *'Get Plan'* nuppu. Näidis plaan on pildil 56:



Pilt 56. Valmis päringu plaan Plan Viewer liideses.

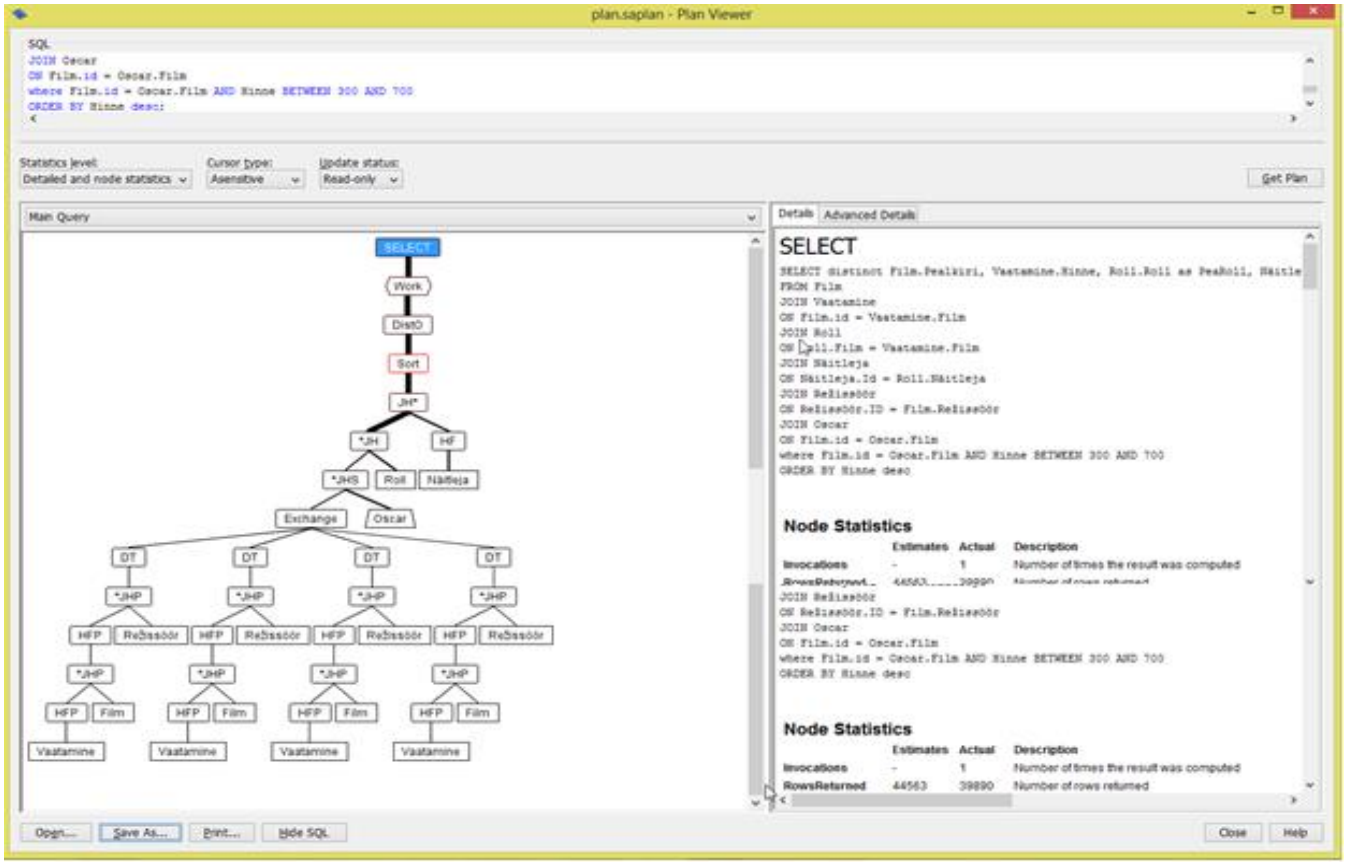
Plaan näitab informatsiooni genereeritud puu kohta. Kuna sisestatud päring on väga lühike ja lihtne, siis koostatud puu sisaldab ainult ühte serva. Antud näide oli mõeldud *Plan Viewer*-i liidesega tutvumiseks. Detailne analüüs on tehtud, kasutades päringut 73.

Päring 73:

```
SELECT distinct Film.Pealkiri, Vaatamine.Hinne, Roll.Roll as PeaRoll,
Näitleja.Eesnimi, Näitleja.Perenimi as Näitleja , Režissöör.Eesnimi
,Režissöör.Perenimi as Režissöör FROM Film
JOIN Vaatamine
ON Film.id = Vaatamine.Film
JOIN Roll
ON Roll.Film = Vaatamine.Film
JOIN Näitleja
ON Näitleja.Id = Roll.Näitleja
JOIN Režissöör
ON Režissöör.ID = Film.Režissöör
JOIN Oscar
ON Film.id = Oscar.Film
where Film.id = Oscar.Film AND Hinne BETWEEN 300 AND 700
ORDER BY Hinne desc;
```

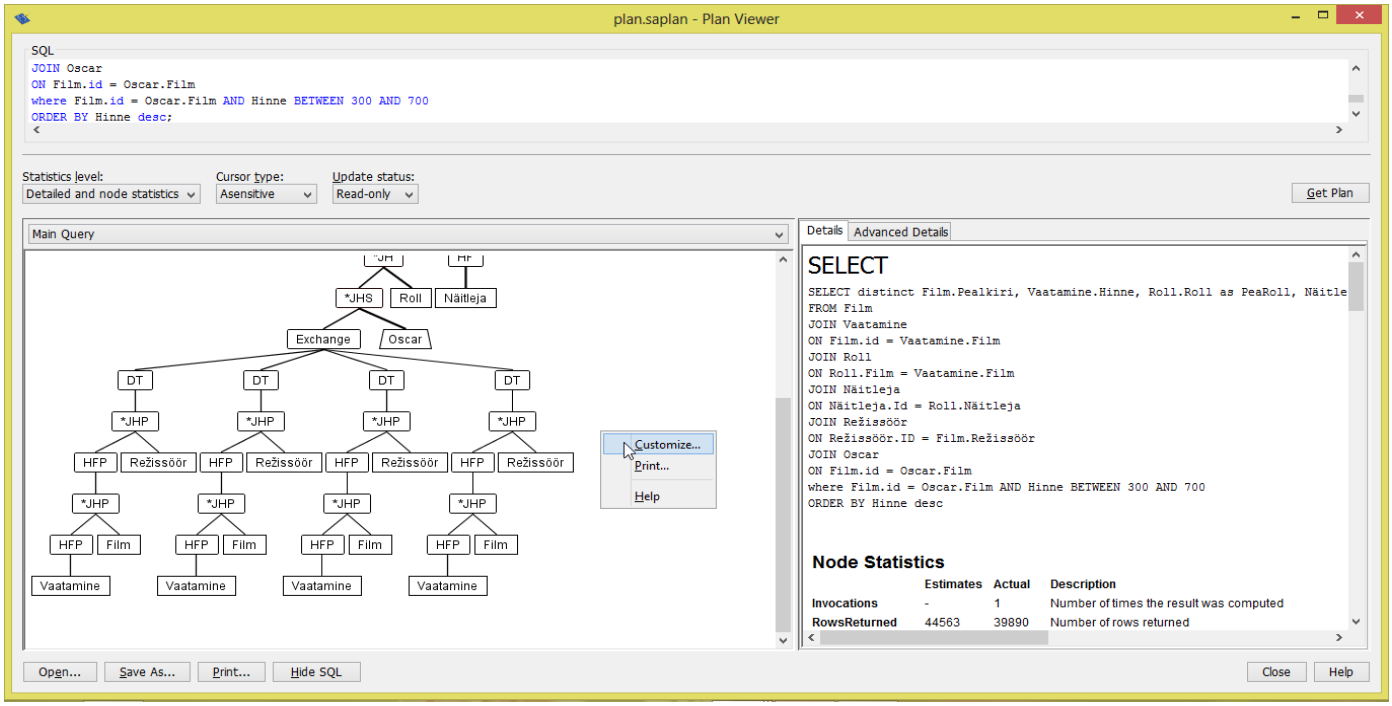
- Execution time: 0.266 seconds

Päringu osad, mis mõjutavad jõudlust negatiivselt on märgitud musta pideva joone või punase piiriga. Näiteks, paksud jooned tippude vahel näitavad, et töödeldud ridade arv on märgatavalt suur. Paks joon üle puu võib tähendada, et on vaja lisada indeks [1]. Antud juhul tabeli „Vaatamine“ lähenemine oli ebaefektiivne, kuna enne oli juba kasutatud sorteerimist, mis aeglustas päringu täitmist. Tabelis 12 on väljatoodud tabelite „Film“, „Vaatamine“, „Oscar“, „Näitleja“, „Roll“ ja „Režissöör“ indeksid. Päringu 73 puu on toodud pildil 57.



Pilt 57. Päringu 75 puu.

Punane piir näitab, et operatsioon on aeganõudev võrreldes teiste operatsioonidega antud plaanis. Tabeli „Vaatamine“ sorteerimine oli antud päringus ebaefektiivne operatsioon, sest oli vaja läbi vaadata kõiki andmeid ning leida tabelis sobiv indeks iga tabeli „Film“ atribuudi „id“ jaoks. Graafilist kuju saab muuta *Interactive SQL -> PlanViewer* keskkonnas. Selleks on vaja teha paremklikk plaani alumises vasakpoolses osas *Plan Viewer*'is ning valida „Kohanda“ (*Customize*). Muudatused rakendatakse järgnevale graafilisele plaanile (Pilt 58).



Pilt 58. Graafilist kuju muutmine.

Tabel 12. Andmebaasi „Film“ tabelite indeksid.

Film	**Näitleja**	**Oscar**	**Vaatamine**	**Režissöör**	**Roll**
IX_Film_Režissöör	IX_Näitleja_Perenimi	Fk_rooli eest	IX_Vaatamine_Hinne	UN_Režissöör_Perenimi	fk_Film_Näitleb
Fk_Lavastab	Näitleja (primaarvõti)	Fk_filmi_eest	Fk_Vaatamine_film	Režissöör (primaarvõti)	
Film (primaarvõti)		Oscar(primaarvõti)	Vaatamine (primaarvõti)		

Kasutaja saab vaadata graafilist plaani, graafilist plaani koos kokkuvõttega või graafilist plaani detailse statistikaga. Graafiline plaan koos statistikaga võimaldab otseselt võrrelda hinnanguid, mis olid tehtud tegeliku statistika alusel ja mis olid kasutatud päringu optimeerija poolt plaani koostamisel. Tuleb meeles pidada, et optimeerija ei saa tihti täpselt hinnata päringu täitmiseaega, nii et võivad esineda erinevused prognoositud ja tegelike väärtuste vahel.

5.3.1 Graafilise plaani statistika analüüs

Graafilist plaani on kasulik kasutada, et leida päringu puu kõige aeglasema lüli. Plaani abil saab kuvada andmebaasi parameetreid ning teisi globaalseid omadusi, mis mõjutavad päringu töötlemist.

Lisaks on võimalik üle vaadata selektiivsuse tulemust. Predikaadi selektiivsus on ridade arv, mis rahuldab päringu tingimusi. Predikaatide hinnanguline selektiivsus näitab, millest lähtub optimeerija kulude prognoosimisel. Täpsed selektiivsuse hinnangud on kriitilised optimeerija korraliku töö tagamiseks. Näiteks, kui optimeerija ekslikult hindab predikaati väga selektiivseks (näiteks, selektiivsusega 5%), kuid tegelikult predikaat on palju väiksema selektiivsusega (näiteks, selektiivsusega 50%), siis jõudlus on häiritud. Kuigi selektiivsuse hinnangud ei pruugi olla täpsed, võib suur hinnangute erinevus tekitada probleemi.
Kui kasutaja ei ole kindel, kuivõrd on selektiivsuse informatsioon täpne, siis tasub kasutada lauset `CREATE STATISTICS`, et genereerida veergudele uut statistikat. Erandina võib kasutada detailset selektiivsuse statistikat, kuid selline lähenemine võib tekitada raskusi. Ebapiisava selektiivsuse indikaatorid võivad esineda järgnevates kohtades [29]:

- *RowsReturned* (nii tegelik, kui ka hinnanguline). *RowsReturned* on ridade arv tulemuses. *RowsReturned* statistika esineb puu juuretipus. Kui hinnanguline ridade arv on tegelikku ridade arvust oluliselt erinev, siis predikaatide selektiivsus on lisatud antud servale või alampuule.

- Predikaatide selektiivsus (nii tegelik, kui ka hinnanguline). Tuleb vaadata predikaadi pealkirja, et saada teada predikaatide selektiivsust. Kui predikaat on baasveerust väljaspool, mille jaoks pole histogrammi, tuleb täita lause `CREATE STATISTICS` histogrammi saamiseks. Kui selektiivsuse viga jääb alles, tasub kaaluda hinnangute täpsustamist koos päringu predikaadiga.
- Hinnangu allikas. Selektiivsuse hinnangute allikas on loetletud predikaadi pealkirjas, „Statistika“ paneelis. Kui allikas on *Guess*, puuduvad optimeerijal andmed, et määrata predikaadi filtri omadusi (näiteks, histogrammi kadumine). Kui allikas on *Index* ning selektiivsuse hinnangud on valed, võib olla probleem on selles, et indeks on tasakaalustamata. Sellisel juhul võib kasuks olla indeksi defragmenteerimine lausega `REORGANIZE TABLE`. Täpsemini sellest räägitakse

punktis 3.12.1. Enne fragmenteerimist tuleb teada, mis andmed seda vajavad ning millised probleemid võivad tekkida.

- *RunTime* ja *FirstRowRunTime* (nii tegelikud, kui ka hinnangulised väärtused juuretipus). *RunTime* võib ilmuda alampuu statistikas, juhul kui see on olemas antud tipu jaoks. *RunTime* tõlgendamine sõltub statistika osast, kuhu see ilmub. *RunTime* on kumulatiivne aeg, mis operaator kulutas selle tipu töötlemiseks. Alampuu statistikas *RunTime* esindab kogu täitmise ajakulu terve alampuu jaoks. Niisiis, enamuse operaatorite jaoks *RunTime* ja *FirstRowRunTime* on sõltumatud parameetrid, mis peavad olema eraldi analüüsitud. *FirstRowRunTime* on aeg, mis on vaja kulutada esimese rea tulemuse valmistamiseks antud tipus. Kui *RunTime* tipu väärtus on suurem, kui oodatud tabeli või indeksi skaneerimisel, siis saab jõudlust parandada `REORGANIZE TABLE` päringu abil (vt. punkt 3.12.1).

Graafilise plaani abil saab üle vaadata vahemälu jõudlust. Kui *CacheRead* parameetri väärtus on sama kui *CacheHits* väärtus, siis töödeldud objektid selle SQL päringu jaoks asuvad vahemälus. Kui *CachRead* on suurem, see tähendada, et andmebaasi server loeb tabeleid ja indekseid kettalt. Mõnedes olukordades nagu hash funktsiooni lisamine, see on oodatav. Muudel juhtudel nagu tsükkel tsükli sees, vähene vähemälu *Hit* parameetri suhe võib viidata sellele, et pole piisavalt palju vahemälu (puhvrit), et lubada päringu täitmist. Selles olukorras tuleb lisada vahemälu.

5.3.2 Vahemälu kasutamine jõudluse parandamiseks

Andmebaasi vahemälu (ingl.k. *cache*) [30] on mälu tüüp, mis on kasutatud serveri poolt andmebaasi lehekülgete salvestamiseks korduvalt kiire juurdepääsu tagamiseks. Mida rohkem lehekülgi on kättesaadavad vahemälust, seda vähem kordi peab andmebaasi server lugema kettalt andmeid. Vahemälu suurus on sageli oluline faktor jõudluse määramisel kuna andmete lugemine kettalt on aeglane operatsioon.

Süsteemide jaoks, mis ei ole pühendunud andmebaasi serveritele, võib vahemälu suuruse piiramine parandada üldist süsteemi jõudlust, jättes teiste protsesside jaoks rohkem mälu. Üldiselt dünaamiline

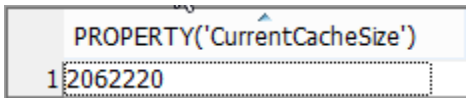
vahemälu suuruse määramine reguleerib vahemälu suurust asjakohaselt ja automaatselt läbi monitooringu süsteemi. Kasutajal on võimalus määrata –c valikuõigust, et kontrollida andmebaasi vahemälu suurust andmebaasi serveri käsurea abil. Andmebaasi serveri sõnumite aken kuvab käivitamisel vahemälu suuruse. Päringu 74 abil võib kasutaja vaadata hetkelist vahemälu suurust.

Päring 74:

```
SELECT PROPERTY ('CurrentCacheSize');
```

- Execution time: 0.016 seconds

Tulemus on toodud pildil 59.



Pilt 59. Pärinu 74 tulemus.

Krüpteeritud andmebaasidel peab olema piisavalt palju vahemälu, et minimeerida I/O operatsioone. Sisend- ja väljundoperatsioonid on krüpteeritud andmebaasides võrdlemisi mahukad, sest krüpteerimist tuleb teha iga tehingu jaoks.

Kokkuvõttes, kui kasutada ABJS-i poolt pakutud optimeerimise võimalusi, kombineerides neid teadmistega andmebaasidest ja SQL päringutest, saab märgatavalt tõsta isegi suure andmebaasi jõudlust ja vähendada vajaliku informatsiooni saamiseks kuluvat aega.

6. Üliõpilaste tagasiside statistika ja analüüs

Antud õppematerjal oli antud tutvumiseks 28 üliõpilasele, kes läbisid kevadel 2016 aine "Andmebaasid". Iga üliõpilane luges neljandiku tööd läbi ning täitis loetud õppematerjali kohta küsimustikku (vt. Lisa 7). Küsimustiku eesmärk oli välja selgitada õppematerjali nõrgad küljed. Järgmiselt on toodud küsimustiku koondtulemused.

1. Materjali esitamisviisi eelised

   Küsimustiku esimeses küsimuses küsiti üliõpilastelt õppimisharjumuste kohta. Nimelt, millist õppematerjali esitamisviisi üliõpilased eelistasid. See oli valikvastustega küsimus; üliõpilased võisid valida järgnevad vastused: tekst, pilt, video või muu. Tulemused on näidatud Tabelis 13. Antud uuring näitas, et vähemalt kolm neljandikku üliõpilastest eelistavad saada õppematerjali teksti või pildi kujul.

   Tabel 13. Üliõpilaste eelised õppematerjali esitamisviiside ja andmebaaside aine kontekstis.

Tekst	Pilt	Videojuhis	Muu
24	21	9	6

2. Materjali kvaliteedi hindamine

   Üliõpilased hindasid erinevaid peatükke erinevalt, Koondtabelis 14 on näidatud üliõpilaste valikud, mille keskmine oli 4.29. Peamisteks puudusteks peeti teksti keerukus ja lausete ülesehitus, mis raskendasid materjalist arusaamist.

   Tabel 14. Materjali kvaliteedi hinne.

Üks	Kaks	Kolm	Neli	Viis
0	0	4	14	10

3. Materjali sisu hindamine

Õppematerjali sisu hindasid tudengid kõige sagedamini hindega neli (vt. Tabel 15). Suures osas on tulemus seotud eelmise küsimuse tulemustega: lausete keerulised sõnastused ja ülesehitus takistasid teksti mõistmist ja arusaamist.

Tabel 15. Materjali kvaliteedi hinne.

Üks	Kaks	Kolm	Neli	Viis
0	0	3	13	12

4. Õppematerjali kitsaskohad

Üliõpilasi paluti välja tuua loetud peatüki kitsaskohad. Antud küsimuses olid ette antud valikvastused. Materjali puudusteks osutusid teksti keerukus ning kohati kohmakas lausete ülesehitus (va. Tabel 16). Teiste kitsaskohtade hulgas olid välja toodud näidete või seletuste puudus ning vormistuse vead.

Tabel 16. Õppematerjali kitsaskohad.

Teksti keerukus	Keeleline struktuur	Näidete puudumine	Seletuste puudumine	Vormistus	Pole kitsaskohti	Küsimus polnud vastatud
7	6	5	4	2	7	6

5. Materjali informatiivsus.

Antud küsimuse eesmärk oli uurida, kuivõrd informatiivne oli õppematerjal üliõpilastele. Enamus (78%) vastasid, et pärast materjali lugemist sai materjali lugeja midagi uut teada SQL keele, andmebaasi või optimeerimise kohta. Viis respondenti ei saanud midagi uut teada ning üks tudeng ei vastanud antud küsimusele. Allpool on toodud vastuste Tabel 17.

Tabel 17. Õppematerjali kasu üliõpilastele.

Jah	Ei	Küsimus pole vastatud
22	5	1

Selle bakalaureusetöö raames koostatud õppematerjal oli tehtud kasutades kahte esitamisviisi - teksti ja pilti. Need esitlusviisid olid ka kõige populaarsemad üliõpilaste seas. Pärast tagasiside analüüsi, on parandatud lausete ülesehitus, vormistus, seletatud mõisted ja lisatud näited.

7. Kokkuvõte

Käesolev bakalaureusetöö on mõeldud üliõpilastele juhendiks *SQL Anywhere 17.0* versiooni kasutamisest ning annab ülevaate SQL päringute optimeerimise võimalustest, päringute koostamise reeglitest, tabelite indekseerimisest ning päringu planeerijast *Plan Viewer*.

SQL on andmete manipuleerimise, mitte programmeerimise keel. Kõiki andmete päringuid tehakse andmebaasis `SELECT`-lausete abil. Päringu kirjutamisel on oluline otsinguparameetrite valik ja kombineerimine, et võimalikult täpselt piiritleda otsimisruumi. Selleks saab kasutada järgmisi tehnikaid:

- *boole*'i loogikaoperaatoreid (`AND`, `OR` või `NOT`) ja sulge;

  Kolmest operaatoritest `AND` osutus kõige ebaefektiivsemaks. Selle põhjuseks on rangem kontroll, mida teostatakse päringu töötlemisel. `OR` operaator on kõige kiirem, sest selle puhul ei pea kehtima kõik tingimused, mis on ühendatud ainult operaatoriga. Piisab, kui üks tingimustest kehtiks. Mida rohkem ridu tagastab päring, seda suurem on selle töötlemise aeg. Seetõttu päringud, kus on kombineeritud mitu operaatorit ning on kasutatud sulge prioriteedi määramiseks võtavad tunduvalt rohkem aega, kui lihtsamate päringute käivitamine. Võrreldes operaatorite tööd sekundites, siis `AND` operaator on kõige aeglasem. Selle operaatoriga võib päring konkreetsetel juhtudel töötada kuni kolm korda aeglasemini, kui näiteks `OR` operaatoriga (vt. Tabel 1). Väga oluline on panna sulud õigesse kohta, sest see mõjutab tulemust. Näiteks, kui sulud on pandud valesti, siis tulemuses võivad olla read, mis ei vasta päringu eesmärgile.

- Fraasiotsing ja regulaaravaldised;

  Mida keerulisem on koostatud regulaaravaldis, seda rohkem aega võtab päringu töötlemine. Seetõttu kasutaja peab hoolikalt läbi mõtlema regulaaravaldise sisu üle, et tulemus ei sisaldaks ebavajalikke ridu. Kui tulemusse sattuvad üleliigsed read, siis päringu töötlemise aeg kasvab. Läbivaadatud regulaaravaldise operaatoritest (`CONTAINS`, `SIMILAR TO` ja `REGEXP`) `SIMILAR TO` osutus kõige aeglasemaks. Teine kõige aeganõudvam operaator on `REGEXP`. `LIKE` operaator on kõige kiirem.`LIKE` operaator võib töötada 8 korda kiiremini kui `SIMILAR TO` operaator. (vt. Tabel 2).

- kärpimine;

  Kärpimise puhul tuleb hoolikalt läbi mõelda, kuhu panna metamärk otsingu mustris. Kui see on pandud valesse kohta, siis on oht, et tulemus sisaldab ebasobivaid ridu või üldse ei sisalda vajalikke ridu, mille tõttu päringu täitmise kiirus suureneb ja produktiivsus langeb. Päring võib tagastada ridu, mis ei vasta püstitatud ülesandele.

Tuleb arvestada sellega, et õigesti valitud SQL päringu komponendid võivad tunduvalt vähendada päringu täitmiseaega. Näiteks, olukordades, kus alampäringu tulemus on väga mahukas (näiteks, 5000 rida), siis `EXISTS` operaator töötab kaks korda kiiremini, kui `IN` operaator. Sama väide kehtib ka vastupidi: kui alampäring ei ole väga mahukas (näiteks, 2000 rida), siis `IN` operaator töötab tunduvalt kiiremini, kui `EXISTS` operaator (vt. Punkt 3.7.1).

Veel üks võimalus tõsta andmebaasi töökiirust on indeksite kasutamine. Indeksid on andmebaasi objektid, mis kiirendavad andmete otsimist andmebaasist ja vähendavad päringu töötlemise aega. Indeksi struktuur võib olla väga erinev ning indeksi loomisel antakse sellele unikaalne nimi, mida koostatakse loodud standartide järgi. Iga indeksi tüüp on loodud konkreetse eesmärgiga ning selleks, et saavutada andmebaasi maksimaalset kiirust, tuleb teada, mis olukorras missugust indeksi tüüpi kasutada. Lisaks tuleb läbi mõelda, mis veerud peavad olema seotud indeksiga. Näiteks, kui koostada päring, mille kõik veerud on seotud sama indeksiga, siis töötlemise aeg suureneb peaaegu kahekordselt (vt. Tabel 9). Kui kasutada indeksit oskuslikult, siis päringu ajakulu võib väheneda kuni 15 korda (vt. Tabel 11).

Produktiivsus sõltub SQL päringu koostamise oskusest. Pikemaid ja keerulisemaid päringuid on raske optimeerida lihtsalt vaadates nende peale. Samuti on raske hinnata, kui optimaalne on koostatud päring. Sellisel juhul on abiks *Plan Viewer*, mille abil saab vaadata jooksutatud päringu detaile alates sellest, kui palju aega võttis päringu käivitamine ja lõpetades päringu puu ning statistikaga. Antud informatsiooni põhjal saab otsustada kas ja kuidas parandada sisestatud päringut.

Optimeerimist ei saa õppida ühe päevaga, see tuleb kogemusega. Teooria on väga mahukas ja kohati päris raskesti arusaadav, kuid kombineerides teoreetilisi teadmisi harjutustega, saab saavutada suurepärase tulemuse nii väikese kui ka suure andmebaasi puhul.

Koostatud materjal on tutvustatud Tartu Ülikooli bakalaureuse esimese aasta informaatika õppekava üliõpilastele. Tagasisidena oli tehtud küsimustik, mille järgi analüüsiti töö aktuaalsust (vt. punkt 6). Tagasiside andis alust arvata, et koostatud õppematerjal on üliõpilastele vajalik uute teadmiste

omandamiseks SQL päringute koostamisest, analüüsimisest ja optimeerimisest. Seega bakalaureusetöös püstitatud eesmärk on saavutatud.

8. Kasutatud kirjanduse loetelu

1. iAnywhere Solutions, Inc. (2010) *Analyzing performance using the graphical plan with statistics.* [Online].
   http://dcx.sybase.com/1200/en/dbusage/ug-optimizer-s-3454485.html
   Viimati vaadatud: 22.03.2017

2. solid IT gmbh. (2017) *DB-Engines ranking – Trend Popularity.* [Online].
   http://db-engines.com/en/ranking_trend
   Viimati vaadatud: 23.03.2017

3. Sander Ott. (2011) Veebiteenuste kasutamine SQL Anywhere andmebaasis. [Online].
   http://dspace.ut.ee/bitstream/handle/10062/32781/thesis.pdf?sequence=1&isAllowed=y
   Viimati vaadatud: 22.03.2017

4. SAP. (2015) Integrated report. [Online].
   https://www.sap.com/docs/download/investors/2016/sap-2016-annual-report-form-20f.pdf
   Viimati vaadatud: 22.03.2017

5. Anne Villems. (2016) Päringute optimeerimine. [Online].
   https://courses.cs.ut.ee/MTAT.03.264/2016_spring/uploads/Main/slaidid%2012
   Viimati vaadatud: 20.03.2017

6. Sybase Inc. (2014) *SAP Adaptive Server Enterprise 16.0.* [Online].
   http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.help.ase.16.0/doc/html/title.html
   Viimati vaadatud: 22.03.2017

7. R.J. Niemiec. (2007) *Oracle Database 10g Perfomance Tuning Tips&Techniques.*

8. Dell Inc. (2016) SQL Optimizer for SAP ASE. [Online].
   http://software.dell.com/products/sql-optimizer-for-sap-ase/
   Viimati vaadatud: 22.03.2017

9. Vilve Seiler ja Kärt Mill. *Päringute koostamine.* [Online].
   https://sisu.ut.ee/otsing/p%C3%A4ringu-koostamine
   Viimati vaadatud: 22.03.2017

10. iAnywhere Solutions, Inc. (2012) *LIKE, REGEXP, and SIMILAR TO search conditions*. [Online].
    http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.sqlanywhere.12.0.1/dbreference/like-regexp-similarto.html
    Viimati vaadatud: 22.03.2017

11. TechOnTheNet.com. *(2003-2016) SQL Server: SELECT TOPP statement.* [Online].
    http://www.techonthenet.com/sql_server/select_top.php
    Viimati vaadatud: 22.03.2017

12. Oracle. (2010) *SQL Tuning Overview.* [Online].
https://docs.oracle.com/cd/B19306_01/server.102/b14211/sql_1016.htm#i30971
Viimati vaadatud: 22.03.2017

13. Sybase Inc. (2012) *SAP Infocenter*. [Online].
http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc01612.0500/doc/html/eli1309270514743.html
Viimati vaadatud: 22.03.2017

14. SQL Join Explained. [Online].
http://www.sql-join.com/
Viimati vaadatud: 22.03.2017

15. Helle Hein. (2010) *Skeemi objektide haldamine.* [Online].
http://kodu.ut.ee/~hhein/abh_2010/Praktikum_4.pdf
Viimati vaadatud: 22.03.2017

16. Einar Pihlap. (2011) *Trigerite kasutamine andmebaasi optimeerimisel.*

17. Ken Milligan. (2013) *TechTip: Simplify and Centralize Your SQL Triggers*. [Online].
http://www.mcpressonline.com/sql/techtip-simplify-and-centralize-your-sql-triggers.html
Viimati vaadatud: 22.03.2017

18. Sybase Inc. (2011) *Using sp_helpconstraint to find a table's constraint information.* [Online].
http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc32300.1570/html/sqlug/X38624.htm
Viimati vaadatud: 22.03.2017

19. Erki Eessaar. (2012) *Andmebaaside projekteerimine.* [Online].
http://maurus.ttu.ee/ained/IDU0220_2012/doc/4/Teema_IDU0220_14_2012.pdf
Viimati vaadatud: 22.03.2017

20. Kendra Little. (2012) *How To Decide if You Should Use Table Partitioning*. [Online].
http://www.brentozar.com/archive/2012/03/how-decide-if-should-use-table-partitioning/
Viimati vaadatud: 22.03.2017

21. iAnywhere Solutions, Inc. (2010) *REORGANIZE TABLE statement*. [Online].
http://dcx.sybase.com/1200/en/dbreference/reorganize-table-statement.html
Viimati vaadatud: 22.03.2017

22. Sybase Inc. (2011). *Segment tutorial.* [Online].
http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc31644.1570/html/sag2/X55628.htm
Viimati vaadatud: 22.03.2017

23. Helle Hein. (2010). *Andmebaasi juhtimine, hoidla struktuurid.* [Online].
http://kodu.ut.ee/~hhein/abh_2010/Praktikum_2.pdf
Viimati vaadatud: 22.03.2017

24. Priit Raspel. (2015) *Indeksid.* [Online].
http://enos.itcollege.ee/~priit/E-kursus%20(I%20245)%20AB-de%20alused/14/14.htm
Viimati vaadatud: 22.03.2017

25. Leo Nelson. (2010) *SQL Index Naming Convention.* [Online].
http://www.leonelson.com/2010/04/03/sql-index-naming-convention/
Viimati vaadatud: 22.03.2017

26. Erki Eessaar. (2012) *Andmebaasi füüsiline disain.* [Online].
http://maurus.ttu.ee/ained/IDU0220_2012/doc/4/Teema_IDU0220_12_2012.pdf
Viimati vaadatud: 22.03.2017

27. Wikipedia. (2015) Database index. [Online].
https://en.wikipedia.org/wiki/Database_index#Clustered
Viimati vaadatud: 22.03.2017

28. Oracle Corporation. (2016) *How MySQL Uses Indexes.* [Oneline].
http://dev.mysql.com/doc/refman/5.7/en/mysql-indexes.html
Viimati vaadatud: 22.03.2017

29. iAnywhere Solutions, Inc. (2010) *Reading graphical plans.* [Online].
http://dcx.sybase.com/1200/en/dbusage/graphical-plans-queryopt.html
Viimati vaadatud: 22.03.2017

30. iAnywhere Solutions, Inc. (2010) *Use the cache to improve performance.* [Online].
http://dcx.sybase.com/1200/en/dbusage/dynamic-performance-newaspen.html
Viimati vaadatud: 22.03.2017

31. Tartu Ülikool (2013) *Sissejuhatus.* [Online].
https://courses.cs.ut.ee/MTAT.03.264/2013_spring/uploads/Main/L01_Sissejuhatus.pdf
Viimati vaadatud: 22.03.2017

32. H. Vallatse. (2016) *e_Teatmik*. [Online].
http://www.vallaste.ee
Viimati vaadatud: 22.03.2017

33. Wikipedia - The Free Encyclopedia. (2016) *B-tree.* [Online].
https://en.wikipedia.org/wiki/B-tree
Viimati vaadatud: 22.03.2017

34. Copyscape (2011) *DSS vs MIS.* [Online].
http://www.differencebetween.com/difference-between-mis-and-vs-dss/
Viimati vaadatud: 22.03.2017

35. Anne Villems. (2013) *Abimaterjal andmebaasi kursuses kasutatud formalismide lugemiseks*. [Online].
https://courses.cs.ut.ee/MTAT.03.264/2016_spring/uploads/Main/tabel%20jm
Viimati vaadatud: 22.03.2017

36. Wikipedia. (2015) *Matser data.* [Online].
https://en.wikipedia.org/wiki/Master_data
Viimati vaadatud: 22.03.2017

37. Datawarehouse4u.Info. (2008-2009) *OLTP vs. OLAP.* [Online].
http://datawarehouse4u.info/OLTP-vs-OLAP.html
Viimati vaadatud: 22.03.2017

38. Eero Ringmäe. (2005) *Arendusprotsess: andmeaida süsteemi rajamine*. [Online].
http://ringmae.com/materjal/Bakalaureusetoo_Eero%20Ringmae_010636_23.08.05.pdf
Viimati vaadatud: 22.03.2017

39. Lauri Võsandi. *Predikaat loogika.* [Online].
http://lauri.xn--vsandi-pxa.com/tub/qaoes/predicate-calculus.html
Viimati vaadatud: 22.03.2017

Lisad

I. Mõisted

Andmebaasi juhtimise süsteem ehk ABJS (*ingl.k.* Database Managment System ehk DBMS**)** - tarkvara, mille abil luuakse ja kasutatakse andmebaase [31].

Andmete segment (*ingl.k*. Data segment) – andmeportsjon, mida saalitakse mällu ja sealt välja [32].

Automaatne optimeerimine (*ingl.k*. Automatic optimization) – optimeerimine ilma inimese kaasabita vastava tarkvara abil.

B-puu (*ingl.k.* B tree) - infotehnoloogias puukujuline andmestruktuur, kus andmed on sorteeritud ning andmeid on lubatud lisada ja eemaldada [33].

Indeks (*ingl.k*. Index) - võtmete (võtmesõnade) loend; iga võti identifitseerib üht kindlat kirjet. Indeksite kasutamine võimaldab kirjeid kiiresti üles leida [32].

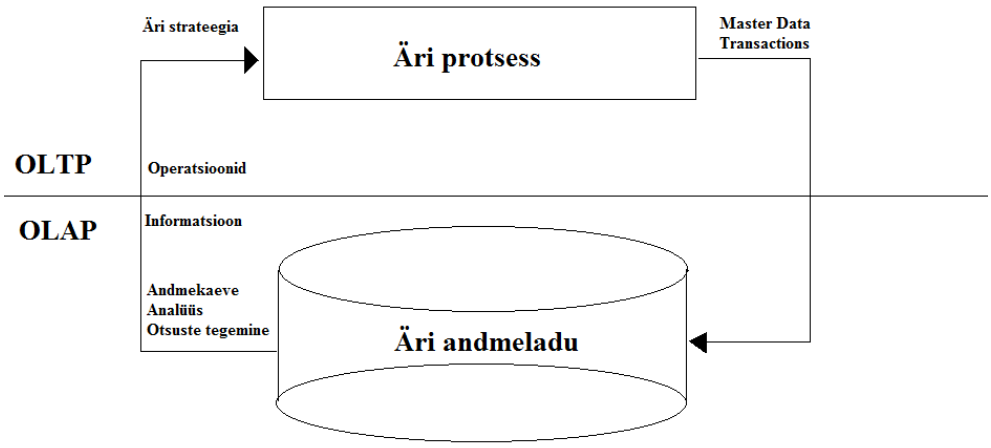
Infosüsteem (*ingl.k.* MIS ehk Management Information System) - informatiivne andmebaasi rakendus, mis annab informatsiooni jõudluse ja andmebaasi kohta [34].

Master Data – äri objektide kokkulepitud esitamine. [36]

Metamärk (*ingl.k.* wildcard) – erisümbol, mida kasutatakse regulaaravaldise koostamisel.

OLTP süsteem (*ingl.k.* OnLine Transaction Processing System) – operatsiooniline infosüsteem/programm/tarkvarapakett, mille ülesandeks on äri igapäevase tegevuse toetamine. [37]

OLAP süsteem (*ingl.k*. OnLine Analytical Processing system) **- s**üsteemid, mis on loodud suure hulga andmete esitamiseks ning suurt hulka kirjeid analüüsivate päringute sooritamiseks. [37]



Pilt 60. OLTP ja OLAP süsteemide erinevus [37].

Puu (*ingl.k.* Tree) - andmestruktuur, kus iga element on ühendatud ühe või mitme temast allpool asuva elemendiga (puu hargneb ülalt alla) [32].

Predikaat (*ingl.k.* predicate) - lause või valem (*formula*), mis sisaldab ühte või mitut muutujat (*variables*). Predikaatlause tõeväärtus oleneb väärtustatud muutujate tõeväärtustest [35].

Relatsioonilise andmebaasi haldur (*ingl.k.* RDBMS ehk Relational Database Management System) – andmebaasi haldur, mis salvestab andmeid omavahel seostatud tabelitena.

Rakenduse elutsükkel (*ingl.k.* Application lifecycle) – tarkvara loomise protsess, mis hõlmab nõuete määratlemist, tarkvara arhitektuuri modellerimist, arendamist ja hooldust.

II. SQL laused tabelite loomiseks

```
create table Režissöör(
id integer not null default autoincrement primary key,
Sünniaeg date not null,
Rahvus varchar(45),
Sugu char(1) not null,
Eesnimi varchar(45) not null,
Perenimi varchar(45) not null
);

create table Näitleja(
id integer not null default autoincrement primary key,
Sünniaeg date not null,
Rahvus varchar(45),
Sugu char(1) not null,
Perenimi varchar(45) not null,
Eesnimi varchar(45) not null
);

create table Film (
id integer not null default autoincrement primary key,
Pealkiri varchar(100) not null,
Aasta integer not null,
Riik varchar(45) not null,
Kestus integer not null,
Keel varchar(45) not null,
Žanr varchar(45) not null,
Režissöör integer,
constraint fk_Lavastab foreign key (Režissöör)
references Režissöör(id)
);

create table Vaatamine (
id integer not null default autoincrement primary key,
Kuupäev date not null,
Hinne integer,
Film integer,
constraint fk_vaatamine_film foreign key (Film)
references Film(id)
);



create table Roll (
Roll varchar(45) not null,
Film integer,
Näitleja integer,
constraint fk_Näitleja_Näitleb foreign key (Näitleja)
references Näitleja(id),
constraint fk_Film_Näitleb foreign key (Film)
references Film(id)
);
```

```
create table Oscar (
Film integer,
Kategooria varchar(45) not null,
Näitleja integer,
constraint fk_filmi_eest foreign key (Film)
references Film(id),
constraint fk_rolli_eest foreign key (Näitleja)
references Näitleja(id)
);
```

III. SQL laused tabelite täitmiseks

i. Tabelid „Režissöör“ ja „Näitleja“

Kuna tabelid „Režissöör“ ja „Näitleja“ on sarnaste atribuutidega, siis nende tabelite täitmise päring on samasugune. Allpool on toodud näidis-päring tabeli „Režissöör“ jaoks.

```
--tabel Režissöör
DECLARE @i INT,
@Eesnimi varchar,
@Perenimi varchar,
@Sünniaeg date,
@Rahvus varchar,
@Sugu char
SET @i = 1
WHILE @i <= 10000 BEGIN
SET @Sünniaeg = DBA.date_rand('1970-09-28','2015-07-06')
SET @Sugu = gender()
-- Start a transaction
IF @i % 10000 = 1 BEGIN
BEGIN TRANSACTION
PRINT 'Transaction started'
END

//print @Sugu
INSERT INTO Režissöör VALUES (@i, @Sünniaeg,
DBA.CreateRandomString(round(rand()*26+1,0)), @Sugu,
DBA.CreateRandomString(round(rand()*26+1,0)),
DBA.CreateRandomString(round(rand()*26+1,0)))
SET @i = @i + 1

IF @i % 10000 = 0 BEGIN
COMMIT
PRINT 'Committed'
END
END
GO
```

Veeru „Sünniaeg“ adnmetüüp on `DATE`. Selle täitmiseks on andmebaasis eraldi tehtud funktsioon `date_rand(@fromDate date, @toDate date)`, mis genereerib juhusliku kuupäeva vahemikus 28.09.1970 ja 06.07.2015. Selle funktsiooni päring on järgmine:

```
CREATE FUNCTION "DBA"."date_rand" ( @fromDate date, @toDate date) returns date
as
begin

declare @days_between int
declare @days_rand int

set @days_between = datediff(day,@fromDate,@toDate)
set @days_rand  = cast(RAND()*10000 as int)  % @days_between
```

```
return dateadd( day, @days_rand, @fromDate )
end
```

Binaarne veerg „Sugu“ saab sisaldada ainult väärtust kas ’M’ või ’F’. Funktsioon `gender()` juhuslikult valib väärtuse:

```
CREATE FUNCTION "DBA"."gender" () returns char
as
begin
declare @gen char
case when round(rand()*26+1,0) % 2 = 1 then set @gen = 'M' else set @gen = 'F'
end
Return @gen
End
```

Veergudes „Nimi“ ja „Rahvus“ oli kasutatud

`DBA.CreateRandomString(round(rand()*26+1,0)` funktsioon, mis genereerib juhuslikku string tüübi väärtuse.

```
CREATE FUNCTION "DBA"."CreateRandomString" (
@passwordLength  smallint )
RETURNS varchar(100)
AS
begin
declare @password varchar(100)
declare @characters varchar(100)
declare @count int
set @characters = ''
-- load up numbers 0 - 9
set @count = 48
while @count <=57
begin
set @characters = @characters + Cast(CHAR(@count) as char(1))
set @count = @count + 1
end
-- load up uppercase letters A - Z
set @count = 65
while @count <=90
begin
set @characters = @characters + Cast(CHAR(@count) as char(1))
set @count = @count + 1
end
-- load up lowercase letters a - z
set @count = 97
while @count <=122
begin
set @characters = @characters + Cast(CHAR(@count) as char(1))
set @count = @count + 1
end
set @count = 0
set @password = ''
while @count < @passwordLength begin set @password = @password +
SUBSTRING(@characters,CAST(CEILING(
(SELECT r FROM RandHelper)*LEN(@characters)) as int),1)
set @count = @count + 1
```

```
end
RETURN @password
End
```

Tabelite täitmise erinevus seiseb selles, et tabelis „Näitleja“ on olemas atribuut „Oscar“, mille lisamine toimub loodud funktsiooni `Oscar()` abil. Päring „Näitleja“ täitmiseks on toodud allpool.

```
DECLARE @i INT,
@Eesnimi varchar,
@Perenimi varchar,
@Sünniaeg date,
@Rahvus varchar,
@Sugu char
SET @i = 1
WHILE @i <= 10000 BEGIN
SET @Sünniaeg = DBA.date_rand('1970-09-28','2015-07-06')
SET @Sugu = gender()
SET @oscar = oscar()
-- Start a transaction
IF @i % 10000 = 1 BEGIN
BEGIN TRANSACTION
PRINT 'Transaction started'
END
//print @Sugu
INSERT INTO Näitleja VALUES (@i@Sünniaeg,
DBA.CreateRandomString(round(rand()*26+1,0)), @Sugu,
DBA.CreateRandomString(round(rand()*26+1,0)),
DBA.CreateRandomString(round(rand()*26+1,0)))
SET @i = @i + 1
-- Commit after each 10,000 row
IF @i % 10000 = 0 BEGIN
COMMIT
PRINT 'Committed'
END
END
GO

CREATE FUNCTION "DBA"."oscar" () returns char
as
begin
declare @oscar char
case when round(rand()*26+1,0) % 2 = 1 then set @oscar = 'Y' else set @oscar =
'N' end
Return @oscar
End
```

ii. Tabel „Film“

```
  -- Run a loop to insert 1,000,000 rows
   DECLARE @i INT,
               @Nimi varchar,
               @Aasta INT,
               @Riik varchar,
```

```
                @Kestus INT,
                @Keel varchar,
                @Žanr varchar,
                @Režissöör INT

  SET @i = 1
  WHILE @i <= 10000 BEGIN
  SET @Aasta = round(rand()*1970+50,0)
  SET @Režissöör = round(rand()*999+1,0)
  SET @Kestus = round(rand()*999+1,0)
    -- Start a transaction
    IF @i % 10000 = 1 BEGIN
       BEGIN TRANSACTION
       PRINT 'Transaction started'
    END
     //print @Sugu
 INSERT INTO Film VALUES (@i, DBA.CreateRandomString(round(rand()*26+1,0)),
 @Aasta, DBA.CreateRandomString(round(rand()*26+1,0)), @Kestus,
 DBA.CreateRandomString(round(rand()*26+1,0)),
 DBA.CreateRandomString(round(rand()*26+1,0)), @Režissöör)
     SET @i = @i + 1
    -- Commit after each 10,000 row
    IF @i % 10000 = 0 BEGIN

      COMMIT
      PRINT 'Committed'
    END
  END
GO
```

iii. Tabel „Oscar“

```
  -- Run a loop to insert 1,000,000 rows
   DECLARE @i INT,
               @Näitleja INT,
               @Film INT
  SET @i = 1
  WHILE @i <= 10000 BEGIN
  SET @Film = round(rand()*999+1,0),
  SET @Näitleja = round(rand()*999+1,0)
    -- Start a transaction
    IF @i % 10000 = 1 BEGIN
       BEGIN TRANSACTION
       PRINT 'Transaction started'
    END
     //print @Sugu
  INSERT INTO Oscar VALUES
 (@Film, DBA.CreateRandomString(round(rand()*26+1,0),@Näitleja))
     SET @i = @i + 1
    -- Commit after each 10,000 row
    IF @i % 10000 = 0 BEGIN
      COMMIT
      PRINT 'Committed'
    END

  END
```

```
GO
```

iv. Tabel „Vaatamine“

```
-- Run a loop to insert 1,000,000 rows
 DECLARE @i INT,
             @Kuupäev date,
             @Hinne INT,
             @Film INT
SET @i = 1
WHILE @i <= 10000 BEGIN
SET @Kuupäev = DBA.date_rand('1970-10-03','2015-07-06')
SET @Hinne = round(rand()*999+1,0)
SET @Film = round(rand()*999+1,0)
  -- Start a transaction
  IF @i % 10000 = 1 BEGIN
     BEGIN TRANSACTION
     PRINT 'Transaction started'
  END

   //print @Sugu
   INSERT INTO Vaatamine VALUES (@i, @Kuupäev, @HInne, @Film)

   SET @i = @i + 1

  -- Commit after each 10,000 row
  IF @i % 10000 = 0 BEGIN
    COMMIT
    PRINT 'Committed'
  END

END

GO
```

IV. Andmebaasi „Film“ olemid, seosed ja relatsioonid

Andmebaasi olemid ja seosed:

Olem Film kajastab filme. Selle tunnusteks on filmi id (primaarvõti), filmi pealkiri (Pealkiri), väljalaskeaasta (Aasta), päritolumaa (Riik), kestus minutites (Kestus), filmi keel (Keel), mis žanrisse film kuulub (Žanr), kes lavastas (Režissöör) ning eelarve (Eelarve). Tunnuseks on ka režissöör (Režissöör), millel on välisvõti.

Olemi Vaatamine kajastab konkreetseid vaatamisi. Selle tunnusteks on vaatamise id (id - on primaarvõti), vaatamise kuupäev (Kuupäev) ja kasutaja hinne filmile (Hinne). Film on välisvõti.

Olem Režissöör on filmide režissööride jaoks ning selle tunnusteks on: isiku id (id - on primaarvõti), eesnimi (Eesnimi), perenimi (Perenimi), sünniaeg (Sünniaeg), rahvus (Rahvus) ja sugu (Sugu).

Olem Näitleja on filmide näitlejate jaoks ning selle tunnusteks on: näitleja id (id - on primaarvõti), eesnimi (Eesnimi), perenimi (Perenimi), sünniaeg (Sünniaeg), rahvus (Rahvus), sugu (Sugu).

Olem Oscar kajastab filmile antud Oscareid. Tunnusteks on filmi id (Film - välisvõti, seotud tabeli Film primaarvõtmega), kategooria, milles auhind on saadud (Kategooria) ja näitleja id (Näitleja on välisvõti, mis on seotud tabeli Näitleja primaarvõtmega).

Olem Roll kajastab rolle, mis näitleja on mänginud. Tunnusteks on rolli nimetus, filmi id (Film - välisvõti, seotud tabeli Film primaarvõtmega) ja näitleja id (Näitleja on välisvõti, mis on seotud tabeli Näitleja primaarvõtmega).

Olemite Film ja Vaatamine vahel on 1:1 seos.

Olemite Režissöör ja Film vahel on 1:n seos Lavastab - üks režissöör võib teha mitu filmi.

Olemite Näitleja ja Film vahel on n:m seos Roll - üks näitleja näitleb paljudes filmides ja ühes filmis on mitmeid näitlejaid. Selle seose atribuudiks on näitleja antud filmis mängitud roll.

Olemite Näitleja ja Oscar vahel on 1:n seos – ühel näitlejal saab olla mitu Oscarit.

Relatsioonid:

Teisendades graafilise mudeli relatsioonideks on saadud järgnevad relatsioonid:

Film(id, Pealkiri, Aasta, Riik, Kestus, Keel, Žanr, Režissöör_id, Eelarve)
Selle relatsiooni funktsionaalsed sõltuvused on:
id → Pealkiri, Aasta, Riik, Kestus, Keel, Žanr, Režissöör_id, Eelarve
Relatsioon on kolmandal normaalkujul.

Vaatamine(id, Kuupäev, Hinne, Film_id)
Selle relatsiooni funktsionaalsed sõltuvused on:
id → Kuupäev, Hinne, Film_id

Kuupäev, Film_id → id, Hinne

Näeme, et relatsioon on kolmandal normaalkujul, sest Kuupäev, Film_id määravad ära id.

Režissöör (id, Eesnimi, Perenimi, Sünniaeg, Rahvus, Sugu)

Selle relatsiooni funktsionaalsed sõltuvused on:

id → Eesnimi, Perenimi, Sünniaeg, Rahvus, Sugu

Relatsioon on kolmandal normaalkujul.

Oscar(Film, Kategooria, Näitleja)

Relatsioon on kolmandal normaalkujul - Film_id, Kategooria määravad üheselt kõik atribuudid, muid funktsionaalseid sõltuvusi pole.

Roll(Roll, Film, Näitleja)

Film_id, Režissöör _id → Roll

Muud funktsionaalsed sõltuvused puuduvad. Tegemist on kolmanda normaalkujuga.

Näitleja(id, Eesnimi, Perenimi, Sünniaeg, Rahvus, Sugu)

Selle relatsiooni funktsionaalsed sõltuvused on:

id → Eesnimi, Perenimi, Sünniaeg, Rahvus, Sugu

Relatsioon on kolmandal normaalkujul.

V. Veeru „Oscar“ lisamine tabelisse „Film“

Tabelisse „Film“ oli töö jooksul lisatud veerg „Oscar“. Selle veeru täitmiseks kasutasin järgmist päringut:

```
ALTER TABLE Film
ADD COLUMN Oscar char(1);
UPDATE
Film
INNER JOIN Roll ON
Film.Id = Roll.Film
INNER JOIN Näitleja ON
Roll.Näitleja = Näitleja.ID
SET
Film.Oscar = Näitleja.Oscar;
```

VI. Veeru „Eelarve“ lisamine tabelisse „Film“

```
alter table Film
add Eelarve integer;

--Veeru taitmine
DECLARE @Eelarve INT,
@Index INT
SET @Index = 1 --- A while loop counter
--- Loop from 1 to 10
WHILE @Index < 10000
BEGIN
SET @Eelarve = Eelarve()
-- Start a transaction
IF @Index % 10000 = 1 BEGIN
BEGIN TRANSACTION
PRINT 'Transaction started'
END

UPDATE
Film
SET
Eelarve = @Eelarve
--WHERE Eelarve = 300
SET @Index = @Index + 1
IF @Index % 10000 = 0 BEGIN
COMMIT
PRINT 'Committed'
END
END
GO

CREATE FUNCTION "DBA"."Eelarve"()
RETURNS INT
AS
BEGIN
DECLARE @Res INT
DECLARE @Lower INT
```

```
DECLARE @Upper INT
---- This will create a random number between 1 and 999
SELECT @Lower = 100 ---- The lowest random number
SELECT @Upper = 1000000 ---- The highest random number
SELECT @Res = ROUND(((@Upper* RAND()   - @Lower* RAND()   -0.1) * RAND() + @Lower),
0)
RETURN @Res
END
```

Teine päring:

```
DECLARE
@Index INT
SET @Index = 1 --- A while loop counter
--- Loop from 1 to 10
WHILE @Index < 10000
BEGIN
-- Start a transaction
IF @Index % 10000 = 1 BEGIN
BEGIN TRANSACTION
PRINT 'Transaction started'
END

UPDATE
Film
SET
Eelarve = (SELECT(ROUND(RAND()*50000+1,0)))
--WHERE Eelarve = 300
SET @Index = @Index + 1
IF @Index % 10000 = 0 BEGIN
COMMIT
PRINT 'Committed'
END
```

VII. Küsitlus

Olen Anastassia Ivanova ja koostasin eestikeelsed õppematerjalid ainele MTAT.03.264 Andmebaasid SQL päringute koostamisest, analüüsimisest ja optimeerimisest *SQL Anywhere 17.0* versiooni näitel. Antud materjal annab tudengitele rohkem võimalusi optimeerimise õppimiseks. Palun Sind neid üle vaadata, lisada materjalidesse kommentaarid ning anda hinnangu-tagasiside. Vastused on anonüümsed ning kasutan neid ainult enda bakalaureusetöös.

1. Millist õppematerjalide esitamisviisi eelistad?

Tekst.

Pilt.

Videojuhis.

Muu_____

2. Hindan materjalide kvaliteeti (näited, pildid, vormistus, terviklikkus) viie palli skaalal järgnevalt:

1(väga halb) 2 3 4 5(väga hea)

Kommentaarid: _____

_____

3. Hindan materjalide sisukust (käsitletud teema sügavus, seletus on piisavalt lihtne ja arusaadavus) viiepalli skaalal järgnevalt:

1(väga halb) 2 3 4 5(väga hea)

Kommentaarid: _____

_____

4. Mida on vaja parandada? _____

_____

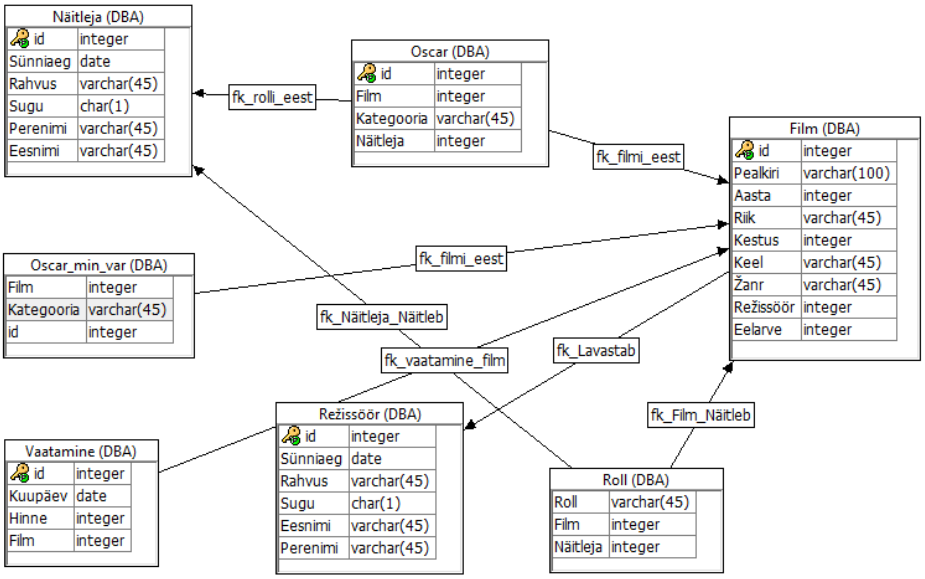
_____

5. Kas said midagi uut teada?

Jah

Ei

Töö praktilise osa jaoks oli loodud andmebaas nimega „Film“, mis sisaldab tabeleid „Režissöör“, „Film“, „Vaatamine“, „Oscar“, „Näitleja“ ja „Roll.



Pilt 61. Andmebaasi „Film“ mudel.

VIII. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Anastassia Ivanova

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
„Päringute koostamine, analüüsimine ja optimeerimine SQL Anywhere 17.0 versiooni näitel“ , mille juhendaja on Ljubov Jaanuska.

1.1 reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi
DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **27.04.2017**