Teema 4. Keerukamad SELECT laused

Sisukord

1. Eesmärk....2
2. Andmebaas....2
3. Relatsioonialgebra operatsioonid "ühend", "lõige" ja "vahe"....4
4. Tabelite ühendamine....9
  4.1 Otsekorrutis....9
  4.2 Ühendamine....10
    4.2.1 Naturaalühendamine....15
    4.2.2 Välisühendamine....16
    4.2.3 Union join....21
    4.2.4 Tabeli ühendamine iseendaga....21
5. Alampäringud....23
  5.1 Võrdluse predikaat....24
  5.2 IN predikaat....26
  5.3 EXISTS predikaat....28
  5.4 Koguselised võrdluse predikaadid....29
    5.4.1 ANY e SOME....30
    5.4.2 ALL....31
  5.5 Unique predikaat....32
  5.6 Match predikaat....32
  5.7 Rohkem kui ühte alampäringut sisaldavad päringud....33
  5.8 Alampäringud mujal kui WHERE klauslis....34
  5.9 Piirangud alampäringutele....35
  5.10 Tabelite võrdlemine....36
6. Grupeerimine....37
  6.1 Kokkuvõttefunktsioon ja tühjad väljad....44
  6.2 Veel kokkuvõttefunktsioone....45
  6.3 Uuem süntaks....45
  6.4 Grupeerimistingimuste lisamine ja kokkuvõttefunktsioonide kasutamine MS Accessi Query Designeris....46
  6.5 Sümmeetriline vahe....49
7. SQL ja statistika....50
  7.1 Mediaan....50
  7.2 Mood....52
  7.3 Keskmise arvutamine ilma suurimat ja väiksemat väärtust arvestamata....52
  7.4 Protsendi arvutamine tervikust....52
  7.5 Kumulatiivne (kumuleeritud) sagedus ja jaotustabel....53
8. SQL SELECT lause täitmise järjekord....55
9. Probleemsetest päringutest ja nende leidmisest....56
10. Relatsioonialgebra ja SQL....59
11. Mõisted....61
12. Kasutatud materjalid....62

1. Eesmärk

- Tutvustada võimalusi hulgateoreetilise summa, hulgateoreetilise vahe ja lõike operatsioonide läbiviimiseks SQLis.
- Tutvustada võimalusi ühendamise operatsiooni läbiviimiseks SQLis.
- Tutvustada alampäringute kasutamist SQLis.
- Tutvustada grupeerimist ning kokkuvõttefunktsioonide kasutamist SQLis.

2. Andmebaas

Esitame järgnevalt SQL-andmebaasi struktuuri kirjelduse.

Osakond (osakonna_nr, osakonna_nimi)
Primaarvõti (osakonna_nr)
Alternatiivvõti (osakonna_nimi)

Tootaja (tootaja_kood, perenimi, aadress, registr_kpv, palk, osakonna_nr)
Primaarvõti(tootaja_kood)
Välisvõti (osakonna_nr) Viitab Osakond (osakonna_nr)

Palgaaste (astme_nr, vahemiku_algus, vahemiku_lopp)
Primaarvõti (astme_nr)

Linn1 (kood, nimi)
Primaarvõti (kood)
Alternatiivvõti (nimi)

Linn2 (kood, nimi)
Primaarvõti (kood)
Alternatiivvõti (nimi)

Tootaja

tootaja_kood	perenimi	aadress	registr_kpv	palk	osakonna_nr
1	Jõgi	Tallinn, Pikk 34	11.11.2001	1400	1
2	Mets	Paide, Roheline 7	12.04.2001	1500	2
3	Kask	Tartu, Tähe 12	10.05.2001	1600	1
4	Triik	Tartu, Kase 12-44	10.03.2001	1800	3
5	Tali	Põlva, Vase 3	10.05.2001	100	

Osakond

osakonna_nr	osakonna_nimi
1	personaliosakond
2	kinnisvaraosakond
3	müügiosakond
4	lepingute osakond

Palgaaste

astme_nr	vahemiku_algus	vahemiku_lopp
1	200	500
2	501	1000
3	1001	3000

Linn1

kood	nimi
0784	Tallinn
0795	Tartu linn
0625	Pärnu linn
0349	Kuressaare linn

Linn2

kood	nimi
0795	Tartu linn
0183	Haapsalu linn
0566	Paide linn
0919	Võru linn

Joonis 1 Näidis andmebaas.

3. Relatsioonialgebra operatsioonid "ühend", "lõige" ja "vahe"

```
<SELECT statement>
{UNION | INTERSECT | EXCEPT} [ALL] [CORRESPONDING [BY [veerg1
[,...]}]]
<SELECT statement>
```

Need aitavad kombineerida mitme SELECT lause tulemusi üheks tulemuseks. Millistele tingimustele peavad need SELECT laused vastama?

- SELECT laused peavad andma tulemuseks võrdse arvu veergusid.
- Kokkulangevad veerud peavad olema samade või ühilduvate andmetüüpidega.
- SELECT laused võivad pöörduda nii ühe sama tabeli kui ka erinevate tabelite poole.

Teiste sõnadega, hulgateoreetiline operatsioon toimub hulkadega, milles on samatüübilised elemendid. Antud juhul on hulkadeks tabelid, elementideks tabelite read ning nõudeks see, et kõikides tabelites peab olema ühesugune ridade struktuur.

Vaikimisi kasutatakse nende operatsioonide läbiviimiseks järgmiseid operaatoreid:

- UNION DISTINCT e UNION – ühendi leidmine,
- INTERSECT DISTINCT e INTERSECT – lõike leidmine,
- EXCEPT DISTINCT e EXCEPT – vahe leidmine.

Nende operaatorite kasutamise korral eemaldatakse operatsiooni tulemusest automaatselt korduvad read, jättes iga korduvate ridade hulga kohta alles vaid ühe rea.

Soovi korral võib kasutada ka operaatoreid, mille kasutamise korral tulemusest korduvaid ridu ei eemaldata.

- UNION **ALL** – ühendi leidmine ilma tulemusest korduvate ridade eemaldamiseta.
- INTERSECT **ALL** – lõike leidmine ilma tulemusest korduvate ridade eemaldamiseta.
- EXCEPT **ALL** – vahe leidmine ilma tulemusest korduvate ridade eemaldamiseta.

Ühendi leidmine	**Vahe leidmine**	**Lõike leidmine**
SELECT perenimi FROM Tootaja WHERE palk < 1500 UNION SELECT perenimi FROM Tootaja WHERE palk > 1900;	SELECT osakonna_nr FROM Osakond EXCEPT SELECT osakonna_nr FROM Tootaja;	SELECT osakonna_nr FROM Osakond INTERSECT SELECT osakonna_nr FROM Tootaja;
Leiab ühendi selliste töötajate perenimedest, kes saavad vähem kui 1500 või rohkem kui 1900 eurot palka. Samaväärne oleks näiteks päring: SELECT tootaja_kood FROM Tootaja WHERE palk>1500 **OR** osakonna_nr=1;	Leiab osakonnad, kus ei tööta mitte ükski töötaja.	Leiab osakonnad, kus töötab vähemalt üks töötaja. Samaväärne oleks näiteks päring: SELECT DISTINCT osakonna_nr FROM Tootaja; Eeldused: • tabelis *Tootaja* on veerul *osakonna_nr* välisvõtme kitsendus, mis tagab, et selles veerus on vaid sellised osakonna numbrid, mis on ka tabelis *Osakond*. • *Tootaja.osakonna_nr* on kohustuslik veerg (NOT NULL). Kitsenduste jõustamine andmebaasis lihtsustab päringute tegemist andmebaasi põhjal!

Kui on määratud fraas CORRESPONDING BY, siis operatsioon tehakse fraasi järgi esitatud veergudel. Eelduseks on, et tabelites (mis moodustuvad SELECT lause tulemusena) on samanimelised veerud.

Kui on määratud sõna CORRESPONDING (ilma sõnata BY), siis operatsioon tehakse *kõigi* veergude põhjal, mis on mõlemas tabelis ühesuguse *nimega.*

Mõnes andmebaasisüsteemis (nt Oracle) kasutatakse *EXCEPT* asemel sõna *MINUS*.

MS Accessis (2019) "INTERSECT" ja "EXCEPT", "CORRESPONDING" ja “CORRESPONDING BY“ kasutada ei saa.

Vaikimisi kasutatakse operaatoreid UNION DISTINCT, INTERSECT DISTINCT, EXCEPT DISTINCT ning operatsiooni tulemusest eemaldatakse korduvad read. See on ka vastavuses relatsioonilise mudeli põhimõtetega, mille kohaselt on relatsioonialgebra operatsiooni tulemus samuti relatsioon

ning relatsioonis ei tohi olla korduvaid korteeže. Read A ja B on ühesugused, kui ridades A ja B olevate järjestatud väärtuste korral a_1, a_2, ..., a_n ning b_1, b_2, ... b_n kehtivad võrdused $a_1=b_1$; $a_2=b_2$,..., $a_n=b_n$
Kui soovite operatsiooni tulemuses säilitada korduvad read, peate kasutama operaatoreid UNION ALL, INTERSECT ALL, EXCEPT ALL.

Näide: Leia töötajad, kelle palk on alla 1500 või üle 1900 euro.

```
SELECT * FROM Tootaja
WHERE palk < 1500
UNION CORRESPONDING BY (tootaja_kood)
SELECT * FROM Tootaja
WHERE palk > 1900;
```

on samaväärne kui:

```
SELECT tootaja_kood FROM Tootaja
WHERE palk < 1500
UNION SELECT tootaja_kood FROM Tootaja
WHERE palk > 1900;
```

Päringu:

```
SELECT * FROM Tootaja WHERE palk < 1500
UNION CORRESPONDING
SELECT * FROM Tootaja WHERE palk > 1900;
```

tulemuseks on tabel, kus on kõik samad veerud kui tabelis *Tootaja.* Ühendamine on toimunud samanimelistel veergudel. Eeliseks on, et SELECT klauslis ei ole vaja veerge kindlas järjekorras kirjutada. See päring on samaväärne kui:

```
SELECT tootaja_kood, perenimi, aadress, registr_kpv, palk, osakonna_nr
FROM Tootaja WHERE palk < 1500
UNION SELECT tootaja_kood, perenimi, aadress, registr_kpv, palk,
osakonna_nr FROM Tootaja WHERE palk > 1900;
```

Päring:

```
SELECT nimi
FROM Linn1
UNION SELECT nimi
FROM Linn2
ORDER BY nimi;
```

annab tulemuse:

nimi
Haapsalu linn
Kuressaare linn
Paide linn
Pärnu linn
Tallinn
Tartu linn
Võru linn

Nagu näete, on kordused elimineeritud.

Päring on ka näide selle kohta, kuidas kasutada UNION lauses sorteerimist, kui alampäringutes leitakse samanimelised veerud.

Päring:

```
SELECT nimi
FROM Linn1
UNION ALL SELECT nimi
FROM Linn2
ORDER BY nimi;
```

aga annab tulemuse

nimi
Haapsalu linn
Kuressaare linn
Paide linn
Pärnu linn
Tallinn
Tartu linn
Tartu linn
Võru linn

Järgnevalt on toodud näiteid tulemuste sorteerimise kohta, kui alampäringute poolt leitakse erineva nimega veerud.

Erineva nimega veerud:

```
SELECT perenimi AS sort_nimi
FROM Tootaja
UNION
SELECT osakonna_nimi AS sort_nimi
FROM Osakond
ORDER BY sort_nimi;
```

NB! UNION DISTINCT, EXCEPT DISTINCT, INTERSECT DISTINCT operatsioonide ***kõrvalefektiks*** *võib* olla, et tulemuseks saadavad read on sorteeritud (et andmebaasisüsteemil oleks lihtsam korduseid eemaldada). Päringute kirjutamisel tuleb taoliste kõrvalefektide ärakasutamisest hoiduda, sest andmebaasisüsteemi uutes versioonides võib olla selline kõrvalefekt kadunud ja SQL laused, mis sellega arvestasid, võivad anda mittesoovitava tulemuse. Seega, kui soovite päringu tulemuses ridu sorteerida, siis kasutage päringus ORDER BY klauslit ja kirjeldage sorteerimise eeskiri.

4. Tabelite ühendamine

Järgnevalt räägitakse tabelite ühendamisest (joinimisest) SQL lausetes.

SQL-andmebaasi põhimõte on see, et andmed on jaotunud erinevate tabelite vahel. Väga sageli tuleb teha päringuid rohkem kui ühe tabeli põhjal.

Kui päringuga on tarvis küsida andmeid mitmest tabelist, kasutatakse ühendamise operatsiooni. Read ühest tabelist on päringus ühendatud teise tabeli ridadega nendes ridades asuvate ühiste väärtuste alusel. Neid operatsioone saab klassifitseerida erinevate kriteeriumite alusel.

Operatsiooni tüüp.

- cross join – otsekorrutis;
- inner join;
  - natural join – naturaalühendamine (inner join ja equijoin ühenimeliste veergude põhjal, tulemuses on korduvad samanimelised veerud eemaldatud nii et jääb vaid üks veerg).
- outer join – välisühendamine;
  - right outer join;
  - left outer join;
  - full outer join.

Osalevad tabelid:

- self join – tabeli ühendamine iseendaga;
- not self join – tabeli ühendamine mõne teise tabeliga.

Kasutatav võrdlusoperaator:

- equijoin;
- non equijoin.

4.1 Otsekorrutis

Kahe tabeli otsekorrutises on kõik read ühe tabeli ridade hulgast ühendatud kõigi ridadega teise tabeli ridade hulgast. Sageli võib korrutise tulemuseks olla väga suur andmehulk. Näiteks kui ühendada otsekorrutise abil kaks tabelit, milles kummaski on 100 rida, on tulemuseks saadavas tabelis
100*100 = 10000 rida.

Korrutise loomist läheb vaja, kui soovite leida olemite paare. Järgnevad päringud on samaväärsed. Mõlemate päringute süntaks on SQL standardi mõttes korrektne.

<u>Näide</u>: Leia kõikvõimalikud töötajate perenimede ja osakonna nimede paarid.

```
SELECT perenimi, osakonna_nimi
FROM Tootaja, Osakond;
```

```
SELECT perenimi, osakonna_nimi
FROM Tootaja CROSS JOIN Osakond;
```

MS Accessis töötab neist päringutest ainult esimene. Teine päring oleks eelistatum, sest toob selgemalt esile päringu tulemuse tähenduse.

4.2 Ühendamine

Equijoini puhul kasutatakse ühendamise tingimuses võrduse kontrolli operaatorit **(=)**.

Non-equijoini puhul on tabelite ridade ühendamine saavutatud teisiti, kui "=" operaatorit kasutades. Sellisel juhul kasutatakse võrdlusoperaatorina: "<", ">", "<=", ">=", "<>", "BETWEEN"

SQLi vana süntaks

Selle süntaksi puhul kirjutatakse tabelite ühendamise tingimused sisse WHERE klauslisse.

```
FROM Tabel1, Tabel2
WHERE Tabel1.veerg1 võrdlusoperaator Tabel2.veerg2
```

Tabel 1. Ühendamist realiseeriva predikaadi (WHERE klauslis oleva atomaarse tingimuse) komponendid

Osa	**Kirjeldus**
Tabel1, Tabel2	Tabelite nimed, milles olevad read ühendatakse.
veerg1, veerg2	Veergude nimed, milles olevate andmete alusel ühendamine toimub. Veerud peavad olema samade või ühilduvate andmetüüpidega (nt üks veerg on tüüp AUTOINCREMENT ja teine LONG INTEGER). Vastasel juhul ei ole võimalik võrrelda nendes veergudes olevaid väärtuseid. Veerud ei pea olema ühesuguse nimega.
võrdlusoperaator	"=","<", ">", "<=", ">=", "<>", "BETWEEN"

Kui erinevates tabelites on ühesuguse nimega veerge, tuleb neile viidata liitnimega **Tabeli_nimi.veeru_nimi**. Vastasel juhul pole see kohustuslik.

Reegel tabelite ridade ühendamiseks (join) päringus: *ühendamise tingimuste (joini tingimuste) minimaalne arv = ühendatavate tabelite arv - 1*

Näide kolme tabeli ühendamise kohta. Nagu näete ühendamise tingimusi on **3-1=2**.

```
SELECT T1.*
FROM Tabel1 AS T1, Tabel2 AS T2, Tabel3 AS T3
WHERE T1.veerg1=T2.veerg1 AND T2.veerg2=T3.veerg1
```

SELECT klauslis esitatakse veergude nimed, millele vastavaid andmeid soovitakse väljundis näha. FROM klauslis loetakse üles ühendatavate tabelite nimed. WHERE klauslis esitatakse tabelite ühendamise tingimus (millistes veergudes olevate andmete kaudu on tabelites olevad andmed omavahel seotud). Väga sageli on ühendamise tingimuses ühelt poolt ühe tabeli primaarvõtme veerg ja teiselt poolt selle tabeliga seotud tabeli välisvõtme veerg.

<u>Näide 1</u> inner join, equijoin (vana süntaks)

Koosta päring, mis esitab töötaja perenime ja osakonna nime, kus ta töötab:

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja, Osakond
WHERE Tootaja.osakonna_nr=Osakond.osakonna_nr;
```

Tulemus:

tootaja_kood	perenimi	osakonna_nimi
1	Jõgi	personaliosakond
2	Mets	kinnisvaraosakond
3	Kask	personaliosakond
4	Triik	müügiosakond

Antud näites ühendatakse tabelis *Osakond* olevad read tabelis *Tootaja* olevate ridadega kasutades veerge *Osakond.osakonna_nr* (tabeli *Osakond* primaarvõtme veerg) ja *Tootaja.osakonna_nr* (tabelis *Tootaja* olev välisvõtme veerg). Nendes veergudes olevate väärtuste kaudu on *Osakond* ja *Tootaja* omavahel seotud.

Kõigepealt loodi kõikvõimalikud töötajate ja osakondade kombinatsioonid (otsekorrutis). Seejärel rakendati vahetulemusele piirangu operatsioon ja jäeti tulemusse vaid sellised read, kus töötaja tabelist võetud reas ja osakonna tabelist võetud reas on ühesugune osakonna number.

Märkus:

```
SELECT *
FROM Tootaja, Osakond
WHERE Tootaja.osakonna_nr=Osakond.osakonna_nr;
```

Kui SELECT klauslis ei määrata soovitavate veergude nimesid, vaid kasutatakse SELECT *, siis SQL tagastab kõigepealt esimese tabeli veerud, seejärel teise tabeli veerud jne. Samanimeliste veergude puhul sõltuvalt andmebaasisüsteemist kas väljastatakse andmed vaid ühest sellisest veerust või eristatakse veeru nimesid tabeli nimede abil.

Ühendamise tingimuses on veergude nimetuses ära märgitud ka tabeli nimi (nt *Tootaja.osakonna_nr*). See on vajalik vaid siis, kui veergude nimed mõlemas tabelis on identsed.

Leia osakonnas nr 1 või 2 töötavate töötajate koodid ja perenimed ning osakonna nimi, kus nad töötavad.

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja, Osakond
WHERE Tootaja.osakonna_nr=Osakond.osakonna_nr AND
Tootaja.osakonna_nr IN (1, 2);
```

SQL standard pakub alates SQL:1992 versioonist ka teistsuguse lausekonstruktsiooni sama ülesande lahendamiseks. Seda soovitatakse eriti kasutada keerulisemate päringute puhul, et eristada ühendamise tingimus ja täiendav piirang. Vana süntaksi kasutamisel võib WHERE klausel muutuda liiga ülekoormatuks.

SQLi uuem süntaks

```
FROM Tabel1 INNER JOIN Tabel2 ON Tabel1.veerg1 võrdlusoperaator
Tabel2.veerg2

SELECT Tabel1.*
FROM (Tabel1 INNER JOIN Tabel2 ON Tabel1.veerg1 = Tabel2.veerg1)
INNER JOIN Tabel3 ON Tabel2.veerg2 = Tabel3.veerg1;
```

<u>Näide 2:</u> Inner join, equijoin (uuem süntaks)

Päringu, mis leiab kõigi töötajate nimed ja nende osakondade nimed, võib esitada ka kujul:

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja INNER JOIN Osakond ON
Tootaja.osakonna_nr=Osakond.osakonna_nr;
```

Leia osakonnas nr 1 või 2 töötavate töötajate koodid ja perenimed ning osakonna nimi, kus nad töötavad.

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja INNER JOIN Osakond ON
Tootaja.osakonna_nr=Osakond.osakonna_nr
WHERE Tootaja.osakonna_nr IN (1, 2);
```

Näide 3: Inner join, equijoin – ühendamine toimub liitvõtme veergude põhjal.

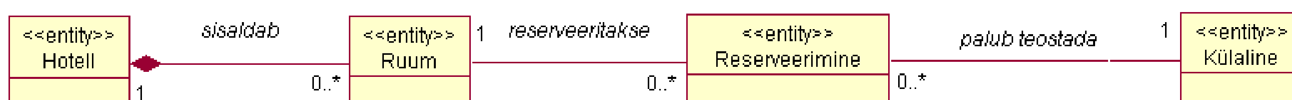
Olgu meil järgmise struktuuriga andmebaas:

```
Hotell(hotelli_nr, nimi, linn)
Primaarvõti(hotelli_nr)

Ruum(ruumi_nr, hotelli_nr, tüüp, hind)
Primaarvõti(hotelli_nr, ruumi_nr)
Välisvõti(hotelli_nr) Viitab Hotell(hotelli_nr)

Külaline(külalise_nr, perenimi, külalise_aadress)
Primaarvõti(külalise_nr)

Reserveerimine(hotelli_nr, ruumi_nr, külalise_nr, alguse_aeg, lopu_aeg)
Primaarvõti(hotelli_nr, ruumi_nr, külalise_nr, alguse_aeg)
Välisvõti(hotelli_nr, ruumi_nr) Viitab Ruum(hotelli_nr, ruumi_nr)
Välisvõti(külaline_nr) Viitab Külaline(külalise_nr)
```



Joonis 2 Liitvõtmetega andmebaasi tabelid.

Järgnev päring leiab iga registreeritud reserveerimise kohta, selle reserveerimise alguse ja lõpu aja, seotud külalise perenime, seotud hotelli nime ja seotud ruumi numbri.

```
SELECT Ru.ruumi_nr, Ru.hind, Ho.nimi, Re.alguse_aeg, Re.lopu_aeg,
Kü.perenimi
FROM Ruum AS Ru, Reserveerimine AS Re, Külaline AS Kü, Hotell AS Ho
WHERE
(Ru.hotelli_nr=Re.hotelli_nr) AND
(Ru.ruumi_nr=Re.ruumi_nr) AND
(Ho.hotelli_nr=Ru.hotelli_nr) AND
(Kü.külalise_nr=Re.külalise_nr);
```

Nagu näete, tekib iga välisvõtmes osaleva veeru kohta üks tabelite ühendamise tingimus. Samaväärne on uuemat süntaksi kasutades tehtud päring:

```
SELECT Ru.ruumi_nr, Ho.nimi, Re.alguse_aeg, Re.lopu_aeg, Kü.perenimi
FROM Külaline AS Kü INNER JOIN (Hotell AS Ho INNER JOIN (Ruum AS Ru
INNER JOIN Reserveerimine AS Re ON (Ru.hotelli_nr = Re.hotelli_nr) AND
(Ru.ruumi_nr = Re.ruumi_nr)) ON Ho.hotelli_nr = Ru.hotelli_nr) ON
Kü.külalise_nr = Re.külalise_nr;
```

Siin on lause teksti lühendamiseks kasutatud *tabeli aliasi* – näiteks tabelile *Ruum* viitamiseks kasutatakse nime *Ru*. Selline nimi kehtib ainult antud SQL lause piirides.

Käesolev näide illustreerib *liitvõtmete* kasutamise probleeme ja eeliseid.

Liitvõtme probleemid.
- Ühendamist sisaldavate päringute keerukus suureneb, sest rohkem ühendamise tingimusi.
- Andmemahtude suurenemine.

Liitvõtme eelised.
- Väheneb vajadus kirjutada ühendamise operatsiooni sisaldavaid päringuid. Näites on nii *ruumi_nr* kui *hotelli_nr* tabelis *Reserveerimine.*

Näide 4: Inner join, non-equijoin

Non-equijoin operatsiooniga on tegemist juhul, kui ridade ühendamine on saavutatud teisiti, kui (=) operaatorit kasutades. Näiteks võib küsida töötaja palgaastet. Palgaastmete tabelis on toodud iga palgaastme jaoks miinimum- ja maksimumpalk. Päringus kasutatakse ridade ühendamiseks BETWEEN predikaati.

```
SELECT tootaja_kood, perenimi, palk, astme_nr
FROM Tootaja, Palgaaste
WHERE palk BETWEEN vahemiku_algus AND vahemiku_lopp;
```

tootaja_kood	perenimi	palk	astme_nr
1	Jõgi	1400	3
2	Mets	1500	3
3	Kask	1600	3
4	Triik	1800	3

Kui töötajale ei leidu vastavat palgaastet, siis tema andmeid päringu tulemuses ei väljastata.

Näide 5: Inner join, non-equijoin

Näites 4 tehtud päring kasutades INNER JOIN süntaksi:

```
SELECT T.tootaja_kood, T.perenimi, T.palk, A.astme_nr
FROM Tootaja AS T INNER JOIN Palgaaste AS A ON (T.palk BETWEEN
A.vahemiku_algus AND A.vahemiku_lopp);
```

Kui soovida päringu tulemuses väljastada ka töötaja, kellel pole sobivat palgaastet kasutada ***outer joini*** e *välisühendamise* operatsiooni.

4.2.1 Naturaalühendamine

Naturaalühendamine on ***equijoini oluline erijuhtum***. Seda operatsiooni saab kasutada juhul, kui ühendamise aluseks olevad veerud on samasuguse nimega. Sellisel juhul ei pea ühendamise tingimust eraldi määrama. Ühendamine toimub kõigi samanimeliste veergude põhjal.

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja NATURAL JOIN Osakond;
```

SQL standard võimaldab kasutada ka järgmist süntaksi:

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja JOIN Osakond USING (osakonna_nr);
```

MS Accessis sellise süntaksiga päringuid teha ei saa.

Need päringud on samaväärsed kui näiteks:

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja, Osakond
WHERE Tootaja.osakonna_nr=Osakond.osakonna_nr;
```

Kui teha päring:

```
SELECT *
FROM Tootaja NATURAL JOIN Osakond;
```

```
SELECT *
FROM Tootaja JOIN Osakond USING (osakonna_nr);
```

siis on tulemuses veerg *osakonna_nr* ainult üks kord.

Kui teha päring:

```
SELECT *
FROM Tootaja INNER JOIN Osakond ON
Tootaja.osakonna_nr=Osakond.osakonna_nr;
```

siis on tulemuses kaks osakondade numbreid sisaldavat veergu (kummastki tabelist) ja veeru nimes on eesliitena tabeli nimi.

Millega tuleb naturaalühendamise kasutamisel arvestada?

1. Ühendamisel kasutatavate veergude puhul peab olema võimalik võrrelda nendes veergudes olevaid väärtuseid (veerud peavad olema samade või ühilduvate tüüpidega).

```
CREATE TABLE A(x INTEGER, y CHAR);
CREATE TABLE B(x CHAR, z CHAR);
SELECT * FROM A NATURAL JOIN B;
```

Päringu käivitamisel annab andmebaasisüsteem selletaolise vastuse:

Syntax error: Can't use <INTEGER> with <CHARACTER> variables in <=> expressions

Süsteemis pole defineeritud operaatorit, mis võimaldaks võrrelda täisarvu tüüpi väärtust tekstitüüpi väärtusega.

2. Ühendatavates tabelites peavad olema sama nimega need ja ainult need veerud, mille alusel soovitakse tabeleid ühendada.

```
CREATE TABLE Liik(liigi_kood SMALLINT PRIMARY KEY,
nimi VARCHAR(30) NOT NULL UNIQUE);

CREATE TABLE Lemmik(kood INTEGER PRIMARY KEY,
nimi VARCHAR(50) NOT NULL,
liigi_kood SMALLINT NOT NULL,
CONSTRAINT fk_liik FOREIGN KEY (liigi_kood) REFERENCES
Liik(liigi_kood));

SELECT * FROM Liik NATURAL JOIN Lemmik;
```

Viimase päringuga ühendatakse tabelid *Liik* ja *Lemmik* kasutades veerge *liigi_kood* ja *nimi.* Probleem on, et tabelis *Liik* on veerus *nimi* liigi nimi (näiteks kodukass, merisiga). Tabelis *Lemmik* on veerus *nimi* lemmiku nimi (näiteks *Miku, Possu*).

Tropashko(2006) märgib, et sageli on andmebaasides paljudes tabelites veerud nagu *kommentaar, loomise_aeg*, *looja.* See muudab naturaalühendamise kasutamise ohtlikuks.

4.2.2 Välisühendamine

Inner join operatsiooni tulemuses on rida juhul, kui kehtib ühendamise tingimus R.a = S.b, kus R.a on väärtus tabeli R veerule a vastavas väljas ning S.b on väärtus tabeli S veerule b vastavas väljas.

Kui rida mingis tabelis ei rahulda ühendamise tingimust, jääb see päringu tulemustest välja. Näiteks kui küsida kõik töötajad ühest tabelist ja nendele vastavaid osakondi teisest tabelist, siis inner joini kasutamise tulemusena ei näe tulemuses andmeid:
- töötajate kohta, kellele pole määratud osakonda,
- osakondade kohta, kus ei tööta ükski töötaja.

Välisühendamise kasutamise tulemusena on võimalik näha ka neid ridu.

Välisühendamise puhul sisaldab tulemus ka:
1. kõik read esimesest operandist (tabelist) – left outer join,
2. kõik read teisest operandist (tabelist) – right outer join,
3. kõik read mõlemast operandist (tabelist) – full outer join.

Left (outer) join süntaks
FROM Tabel1 LEFT OUTER JOIN Tabel2 ON Tabel1.veerg1 võrdlusoperaator Tabel2.veerg2

Sõna OUTER on valikuline, st seda võib kuid ei pruugi kasutada.

Valitakse kõik read tabelist *Tabel1*. Nimi *Tabel1* asub vasakul (*left*) pool fraasi "LEFT JOIN".

<u>Näide 6:</u> left outer join, equijoin

Ühenda töötajate ja osakondade andmed. Näita tulemuses ka selliste osakondade nimesid, milles ei tööta mitte ühtegi töötajat.

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Osakond LEFT OUTER JOIN Tootaja
ON Osakond.osakonna_nr=Tootaja.osakonna_nr;
```

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Osakond LEFT JOIN Tootaja
ON Osakond.osakonna_nr=Tootaja.osakonna_nr;
```

Tulemusse valitakse kõigi osakondade andmed. Kui osakonnas ei ole ühtegi töötajat, siis ühendatakse selle osakonna andmeid esitav rida vaikimisi töötaja reaga *kus kõigi väljade väärtused on vaikimisi väärtused* või nende puudumisel üldse tühjad *(paljudes andmebaasisüsteemides on seal alati tühjad väljad)*. Ülaltoodu päringu tulemus võiks OUTER JOIN-i korral olla näiteks:

Tulemus:

tootaja_kood	perenimi	***osakonna_nimi***
1	Jõgi	personaliosakond
2	Mets	kinnisvaraosakond
3	Kask	personaliosakond
4	Triik	müügiosakond
		lepingute osakond

Meie näites võetakse kõik read tabelist *Osakond*. Lepingute osakonnas ei tööta ühtegi töötajat. Ometi näidatakse selle osakonna nime päringu tulemuses (kollasega tähistatud rida). Kuna ei ole temaga seotud töötajaid, siis reas on töötaja perenime ja koodi väljad tühjad (seal väljades on NULL-markerid).

Probleem – Praegu seda tulemust vaadates jääb mulje, et andmebaasis on andmed viie töötaja kohta, kellest ühe puhul pole teada koodi ja perenime.

Päringu tulemust vaadates ei selgu, kas väljas puudub väärtus sellepärast et andmebaasis on salvestatud rida, kus mõnel väljal ei ole väärtust või jäi see tühjaks välisühendamise kasutamise tulemusel.

```
Right (outer) join süntaks
FROM Tabel1 RIGHT JOIN Tabel2 ON Tabel1.veerg1 võrdlusoperaator
Tabel2.veerg2
```

Võetakse kõik read tabelist *Tabel2*. Nimi *Tabel2* asub paremal pool fraasi "RIGHT JOIN".

Näide 7: right outer join, equijoin

Ühenda töötajate ja osakondade andmed. Näita tulemuses ka selliste töötajate kood ja perenimi, kelle puhul ei ole määratud seotud osakonda.

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Osakond RIGHT JOIN Tootaja ON
Osakond.osakonna_nr=Tootaja.osakonna_nr;
```

Meie näites võetakse kõik read tabelist *Tootaja*. Eksisteerib üks töötaja, kellel pole määratud osakonda. Selle rea puhul on tulemuses osakonna nimi määramata.

Tulemus:

tootaja_kood	***perenimi***	osakonna_nimi
1	Jõgi	personaliosakond
2	Mets	kinnisvaraosakond
3	Kask	personaliosakond
4	Triik	müügiosakond
5	Tali	

Sama tulemuse annab ka ka päring:

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja LEFT JOIN Osakond ON
Osakond.osakonna_nr=Tootaja.osakonna_nr;
```

NB! Sellise ülesande võib lahendada ka teisel viisil. Eeliseks on, et tulemuseks olevas tabelis on kõikides väljades väärtus olemas (väljades ei ole NULLe).

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja INNER JOIN Osakond ON
Osakond.osakonna_nr=Tootaja.osakonna_nr
UNION SELECT tootaja_kood, perenimi, 'puudub' AS osakonna_nimi
FROM Tootaja WHERE osakonna_nr IS NULL;
```

Tulemus:

tootaja_kood	perenimi	osakonna_nimi
1	Jõgi	personaliosakond
2	Mets	kinnisvaraosakond
3	Kask	personaliosakond
4	Triik	müügiosakond
5	Tali	**puudub**

Päringus leitakse selliste töötajate andmed, kellel on seotud osakond ning ühendatakse need selliste töötajate andmetega, kellel seotud osakond puudub.

Näide 8: full outer join, equijoin

Ühenda töötajate ja osakondade andmed. Näita tulemuses ka selliste töötajate kood ja perenimi, kelle puhul ei ole määratud seotud osakonda. Näita tulemuses ka selliste osakondade nimesid, milles ei tööta mitte ühtegi töötajat. Ülesande lahendamiseks võib kasutada täieliku välisühendamise operatsiooni.

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Osakond FULL JOIN Tootaja ON
Osakond.osakonna_nr=Tootaja.osakonna_nr;
```

Tulemus:

tootaja_kood	perenimi	osakonna_nimi
1	Jõgi	personaliosakond
2	Mets	kinnisvaraosakond
3	Kask	personaliosakond
4	Triik	müügiosakond
		lepingute osakond
5	Tali	

Näide 9: left outer join, non-equijoin

```
SELECT T.tootaja_kood, T.perenimi, T.palk, A.astme_nr
FROM Tootaja AS T LEFT JOIN Palgaaste AS A ON (T.palk BETWEEN
A.vahemiku_algus AND A.vahemiku_lopp);
```

Väljastab töötaja koodi, perenime ja palga ka siis, kui tema palk ei mahu tabelis *Palgaaste* esitatud palgaastmestikku.

tootaja_kood	**perenimi**	**palk**	**astme_nr**
1	Jõgi	1400	3
2	Mets	1500	3
3	Kask	1600	3
4	Triik	1800	3
5	Tali	100	

Märkus:
Ajalooliselt on mõnedes andmebaasisüsteemides (nt Oracle) välisühendamise realiseerimiseks kasutatud järgmist süntaksi:

```
SELECT tootaja_kood, perenimi, osakonna_nimi
FROM Tootaja, Osakond
WHERE Tootaja.osakonna_nr (+)=Osakond.osakonna_nr;
```

<u>Näide 10:</u> left outer join, equijoin

Oletame, et tabelis *Tootaja* ei ole veeru *osakonna_nr* korral jõustatud välisvõtme kitsendust. Tänu sellele võivad selles veerus olla väärtused, mida ei ole tabelis *Osakond* veerus *osakonna_nr*.

Tootaja

tootaja_kood	perenimi	aadress	registr_kpv	palk	osakonna_nr
1	Jõgi	Tallinn, Pikk 34	11.11.2001	1400	1
2	Mets	Paide, Roheline 7	12.04.2001	1500	2
3	Kask	Tartu, Tähe 12	10.05.2001	1600	1
4	Triik	Tartu, Kase 12-44	10.03.2001	1800	3
5	Tali	Põlva, Vase 3	10.05.2001	100	
6	Soo	Tartu Kastani 2-14	2.03.2012	1000	10

Osakond

osakonna_nr	osakonna_nimi
1	personaliosakond
2	kinnisvaraosakond
3	müügiosakond
4	lepingute osakond

Järgnevalt on esitatud päring, mis leiab töötajad, kellel on määratud osakonna number, kuid tabelis *Osakond* pole sellist numbrit.

```
SELECT Tootaja.tootaja_kood, Tootaja.osakonna_nr
FROM Tootaja LEFT JOIN Osakond ON Tootaja.osakonna_nr =
Osakond.osakonna_nr
WHERE Osakond.osakonna_nr IS NULL AND
Tootaja.osakonna_nr IS NOT NULL;
```

Tulemus:

tootaja_kood	osakonna_nr
6	10

Pange tähele, et kui andmebaasis oleks deklareeritud välisvõtme kitsendus, siis poleks vaja sellist *kvaliteedikontrolli päringut* regulaarselt käivitada!

Moraal – andmebaasis kitsenduste deklareerimine aitab parandada andmete kvaliteeti ning lihtsustab andmebaasi kasutavate programmide loomist.

4.2.3 Union join

```
<tabeli avaldis 1>UNION JOIN<tabeli avaldis 2>
SELECT Osakond UNION JOIN Palgaaste;
```

osakonna_nr	osakonna_nimi	astme_nr	vahemiku_ algus	vahemiku_ lopp
1	personaliosakond			
2	kinnisvaraosakond			
3	müügiosakond			
4	lepingute osakond			
		1	200	500
		2	501	1000
		3	1001	3000

SQL:1999 standard defineeris UNION JOIN operatsiooni. Date (2003, lk 594) sõnul on plaanis see SQL:2003-st välja jätta ning niimoodi on tõepoolest juhtunud (Melton, 2003). SQL:2003 spetsifikatsioonis kirjutatakse: "In ISO/IEC 9075-2:1999, <joined table> contained a UNION JOIN alternative. This edition of ISO/IEC 9075 removes that alternative."

See näide demonstreerib, et keele (ja seda kirjeldava standardi) arenedes on sealt võimalik ka lausekonstruktsioone eemaldada. See on võimalik vaid juhul, kui nimetatud konstruktsioon pole laialdast kasutamist leidnud.

4.2.4 Tabeli ühendamine iseendaga

Ingl *joining table to itself* e *self join*

Self join operatsioon ühendab tabelis *T* olevad read tabelis *T* olevate ridadega. Tabelite aliasi (*correlation names*) kasutades on võimalik ühendada päringus ridu samast tabelist nii, nagu oleks tegemist kahe erineva tabeliga. Päringu tulemuses on teatud read tabelist ühendatud selle sama tabeli teatud ridadega.

Järgnev päring leiab kõigi selliste töötajate nimed, kes saavad palka rohkem, kui töötaja, kelle kood=1.

Variant 1:

```
SELECT T2.tootaja_kood, T2.perenimi, T2.palk
FROM Tootaja AS T1, Tootaja AS T2
WHERE T1.tootaja_kood=1 AND
T2.palk>T1.palk;
```

Variant 2:

```
SELECT T2.tootaja_kood, T2.perenimi, T2.palk
FROM Tootaja AS T1 INNER JOIN Tootaja AS T2 ON T2.palk>T1.palk
WHERE T1.tootaja_kood=1;
```

Sellist iseendaga ühendamist võib vaja minna, kui ühes päringus on vaja ühte ja sama tabelit pärida tegelikult mitu korda.

Järgnev on nende päringute väga mitteformaalne kirjeldus. Nagu näete, kasutatakse päringus tabeli aliasi (T1 ja T2). Nüüd on tegemist otsekui kahe erineva tabeliga (T1 ja T2), millest esimest saab kasutada töötaja koodiga “1” palga leidmiseks ja teist temas kõrgemapalgaliste töötajate leidmiseks. Kuna seda tehakse ühe SELECT lause sees, saab kõik vajalikud tingimused nüüd esitada selle SELECT lause WHERE klauslis. Päringu tulemus võetakse kahest erinevast virtuaalsest tabelist, mis tegelikult on ühe tabeli niiöelda peegeldused. Virtuaalsete tabelite erinevuse määravad WHERE klauslis olevad tingimused selle virtuaaltabeli kohta. Nii on tabelis T1 ainult üks rida (töötaja kohta, kelle kood=1). Seetõttu piisab teiseks tingimuseks T2.palk>T1.palk, mis võrdlebki töötajate palku valitud töötaja palgaga.

Täpsemalt öeldes ongi T1 ja T2 näol tegemist kahe erineva *muutjaga* (ingl *range variable*). Nende muutujate nimed on T1 ja T2 ning nende muutujate väärtusteks saavad üksteise järel tabelis *Tootaja* olevad read.

Järgneva päringu tulemuseks on osakondade nimede paarid, kusjuures ükski paar ei tohi tulemuses esineda rohkem kui üks korda.

```
SELECT O1.osakonna_nimi AS osakond1, O2.osakonna_nimi AS osakond2
FROM Osakond AS O1, Osakond AS O2
WHERE O1.osakonna_nimi<O2.osakonna_nimi;
```

5. Alampäringud

Alampäring on päring (SELECT lause), mis asub teise SQL-lause sees. Alampäring peab olema sulgudes. Süntaks on sama, mis tavalisel päringul, kuid puudub ORDER BY klausel. Alampäringute liigitus.
Alam- ja põhipäringu seose järgi.

- Mittekorreleeruv alampäring.
  - Alampäringu saab käivitada eraldiseisva lausena.
  - Süsteem *võib* alampäringu täita ühe korra ja kasutada saadud vahetulemusi.
  - Kuna selline alampäring on kasulik ka iseseisvana, siis saab tõmmata paralleele *Matrjoška printsiibiga* tarkvaraarenduses (Gagne, 2011). Süsteem on kihiline, iga kiht teeb seda, mida kõige paremini oskab (sellel on oma "teema"). Iga kiht pakub liidese, mille kaudu kihti kasutada. Süsteemi tuleks arendada seestpoolt väljapoole. Kui kihil on hästi defineeritud liides, siis saab sinna peale ehitada teenuseid, mida arenduse alguses ette ei nähtud.
- Korreleeruv alampäring.
  - Alampäring ja põhipäring on läbipõimunud – alampäringus on viide välisele tabelile – tabelile, mis nimetatakse põhipäringu FROM klauslis.
  - Alampäringut ei saa eraldiseisva lausena käivitada.

Alampäring võib sisalduda nii SELECT lauses, kui ka andmebaasis andmete muutmiseks mõeldud lausetes.

Liigitus tulemuseks oleva tabeli omaduste järgi.

- *Skalaarne alampäring.*
  - Tulemuseks olevas tabelis üks veerg ja null või üks rida. Teisisõnu, tulemuse aste 1 ning võimsus 0 või 1.
  - Alampäringu deklareeritud tüüp on samasugune nagu tulemuseks olevas tabelis oleval ainsal veerul – alampäringu tulemus teisendatakse skalaarseks väärtuseks.
  - Kui skalaarse alampäringu tulemuses on null rida, siis teisendatakse see tulemus NULLiks.
  - Saab kasutada skalaarse konstandi asemel.
- *Rea tagastav alampäring.*
  - Tulemuseks olevas tabelis rohkem kui üks veerg ja null või üks rida. Teisisõnu, tulemuse aste>1 ning võimsus 0 või 1.
  - Alampäringu deklareeritud tüüp on reatüüp – alampäringu tulemus teisendatakse reatüüpi väärtuseks.
  - Kui rea tagastava alampäringu tulemuses on 0 rida, siis teisendatakse alampäringu tulemus reaks kus kõikides väljades on NULLid.
- *Tabeli tagastav alampäring.*
  - Tulemuses: aste>1 ning võimsus 0 või rohkem.

5.1 Võrdluse predikaat

Näide: Leida töötajast, kelle kood on 1, rohkem palka saavad töötajad.

Antud näite puhul on tegemist mittekorreleeruva alampäringuga, mida täidetakse ainult üks kord.

```
SELECT tootaja_kood, perenimi, palk
FROM Tootaja
WHERE palk >(SELECT palk FROM Tootaja WHERE tootaja_kood =1);
```

Sinisega on esitatud põhipäring e peapäring e väline päring ja punasega alampäring.

Mittekorreleeruva alampäringu tunneb ära selle järgi, et selle võib käivitada eraldiseisva SQL lausena.

Põhipäringu WHERE klauslis saab võrdlusoperaatoreid "<","<>","<" ,">", ">=", "<=" (ilma täiendita ANY, ALL või SOME) kasutada vaid siis, kui alampäring on skalaarne (st. tulemuseks olevas tabelis on üks rida ja veerg).

Päring töötab ka sellisena:

```
SELECT tootaja_kood, perenimi, palk
FROM Tootaja
WHERE (SELECT palk FROM Tootaja WHERE tootaja_kood=1)<palk;
```

NB! Selline päring ei töötaks, kui alampäring tagastaks rohkem kui ühe rea, sest võrdlusoperaator "<" on mõeldud kahe skalaarse väärtuse võrdlemiseks, mitte aga skalaarse väärtuse võrdlemiseks väärtuste hulgaga (mis on ka väärtus). Kuna praegu toimub alampäringus töötaja otsing töötaja unikaalse koodi järgi, siis on ka tagatud, et alampäring tagastab maksimaalselt ühe rea.

Näide: Leia keskmisest rohkem palka saavad töötajad.

```
SELECT tootaja_kood, perenimi, palk
FROM Tootaja
 WHERE palk >(SELECT Avg(palk) AS keskmine FROM Tootaja);
```

SQL lause täitmise käigus koostab andmebaasisüsteem ***täitmisplaani***, kus on määratud tulemuseni jõudmiseks vajalikud tegevused ja nende täitmise järjekord. Eelneva SQL lause *võimaliku täitmisplaani* pseudokood:

```
SELECT Avg(palk) AS keskmine INTO kesk FROM Tootaja;
FOR EACH Tootaja
      plk:= Tootaja.palk;
       IF plk>kesk THEN
           Väljasta: Tootaja.tootaja_kood, Tootaja.perenimi, Tootaja.palk
       END IF;
END; /* kesk, plk – muutujad*/
```

Antud näite puhul täidetakse alampäring ainult üks kord (leitakse töötajate keskmine palk).

Näide: Leia nende töötajate perenimed, kes saavad rohkem palka kui nende osakonna töötajad keskmiselt. Antud ülesande lahendamiseks tuleb kasutada korreleeruvat alampäringut.

```
SELECT tootaja_kood, perenimi, palk
FROM Tootaja
WHERE palk >(SELECT Avg(palk) AS keskmine FROM Tootaja AS T2
WHERE T2.osakonna_nr = Tootaja.osakonna_nr);
```

Eelneva SQL lause võimaliku *täitmisplaani* pseudokood:

```
FOR EACH Tootaja
  o_nr:=Tootaja.osakonna_nr;
  plk:= Tootaja.palk;
  SELECT Avg(palk) AS keskmine INTO kesk FROM Tootaja WHERE
osakonna_nr =o_nr;
      IF plk>kesk THEN
         Väljasta: Tootaja.tootaja_kood, Tootaja.perenimi, Tootaja.palk
      END IF;
END;
```

Korreleeruva alampäringu korral nõuab alampäring infot peapäringu ridade kohta ja andmebaasisüsteem täidab seda uuesti iga peapäringu leitava rea korral.

Korreleeruva alampäringu puhul viitab alampäringu WHERE klausel pealauses kasutatavale veerule. **Need põhipäring ja alampäring on üksteisest sõltuvad päringud ja ei saa käivituda iseseisvalt.**

Igale välimise SELECT lause poolt leitud rea kohta pöördutakse uuesti alampäringu poole. Antud juhul pöördutakse nii pea kui ka alampäringus sama tabeli poole – *Tootaja*. Seetõttu tuleb kas pea- või alampäringus kasutada tabeli *Tootaja* aliast. Antud juhul on alampäringus antud tabelile *Tootaja* alias *T2*.

```
SELECT perenimi
FROM Tootaja
WHERE palk >(SELECT Avg(palk) AS keskmine FROM Tootaja AS T2
WHERE T2.osakonna_nr = Tootaja.osakonna_nr);
```

Päringu täitmist võib ette kujutada järgnevalt: iga tabelis *Tootaja* sisalduva rea korral käivitub alampäring, mis leiab sellise osakonna töötajate keskmise palga, mille osakonna number on võrdne selle osakonna numbriga, kus antud töötaja töötab (võrdlus **T2.osakonna_nr = Tootaja.osakonna_nr**). Leitud väärtust võrreldakse antud töötaja palgaga.

Seega kontrollitakse iga töötaja puhul, kas tema palk on suurem, kui tema osakonda kuuluvate töötajate keskmine palk. Kui tema palk on suurem, siis osutub õigeks võrdlus:

```
palk >(SELECT AVG(palk) AS keskmine FROM Tootaja AS T2 WHERE
T2.osakonna_nr = Tootaja.osakonna_nr)
```

ja antud töötaja kohta käiv rida valitakse päringu tulemusse. Siis võetakse ette järgmine töötaja, käivitatakse alampäring uuesti jne. Seega peab alampäring käivituma niipalju kordi, kui on peapäringu poolt leitud ridu. Alati, kui alampäring peab iga peapäringu rea kohta tagastama erinevaid väärtusi, tuleb kasutada peapäringuga korreleeruvat alampäringut.

Näide: Leia iga töötajaid omava osakonna kõige varem registreeritud töötaja. Väljasta osakonna kood, töötaja kood ja töötaja perenimi.

```
SELECT osakonna_nr, tootaja_kood, perenimi
FROM Tootaja
WHERE registr_kpv=(SELECT Min(registr_kpv) AS vanim FROM Tootaja AS
T2 WHERE T2.osakonna_nr= Tootaja.osakonna_nr);
```

Iga tabelist *Tootaja* leitud rea kohta täidetakse alampäring, et leida milline on selle töötajaga samas osakonnas töötavate töötajate seas kõige väiksem registreerimise kuupäev. Leitud kuupäeva võrreldakse vaadeldava töötaja registreerimise kuupäevaga. Kui need ajad on ühesugused, siis järelikult võib töötaja tulemusse valida.

5.2 IN predikaat

```
<in predikaat>::=<rea väärtuse konstruktor> [NOT] IN <in predikaadi väärtus>

<in predikaadi väärtus>::=<alampäring> | (<väärtuste nimekiri>)

<väärtuste nimekiri>::=<rea väärtuse avaldis> (<,> <rea väärtuse avaldis>...)
```

IN predikaadi abil kontrollitakse väärtuse sisaldumist väärtuste hulgas. Väärtuste hulk võib olla päringusse literaalidena sisse kirjutatud (vt teema 3) või selle võib leida kasutades alampäringut.

Alampäring võib küsida andmeid teistest tabelitest kui väline (*outer*) päring.

Näide: Leida osakonnad, kus on vähemalt üks töötaja. Väljasta andmed kõigist tabeli *Osakond* veergudest.

```
SELECT *
FROM Osakond
WHERE osakonna_nr IN (SELECT osakonna_nr FROM Tootaja
WHERE osakonna_nr IS NOT NULL);
```

Ülesande lahendamiseks tuleb läbi viia *semijoin* operatsioon.

Alampäring leiab tabeli *Tootaja* põhjal kõigi osakondade numbrid, kus keegi töötab. Põhipäringu predikaadis kontrollitakse, kas osakonna number sisaldub

alampäringu poolt leitud osakondade numbrite hulgas. Kui sisaldub, siis kuuluvad selle osakonna andmed väljastamisele.

Näide: Leida osakonnad, kus ei tööta ükski töötaja. (Mõnikord nimetatakse sellist päringut ka *anti-joiniks*). Väljasta andmed kõigist tabeli *Osakond* veergudest.

```
SELECT *
FROM Osakond
WHERE osakonna_nr NOT IN (SELECT osakonna_nr
        FROM Tootaja
        WHERE osakonna_nr IS NOT NULL);
```

Ülesande lahendamiseks tuleb läbi viia *semidifference* operatsioon.

Alampäring leiab tabeli *Tootaja* põhjal kõigi osakondade numbrid, kus keegi töötab. Põhipäringu predikaadi abil kontrollitakse, kas osakonna number *ei sisaldu* alampäringu poolt leitud osakondade numbrite hulgas. Kui ei sisaldu, siis kuuluvad selle osakonna andmed väljastamisele.

Kuidas mõjutaksid selle päringu täitmist alampäringu poolt leitavad NULLid?

Oletame, et alampäring leiab järgneva *osakonna_nr* hulga (1, 2, 1, 3, NULL).

Põhipäring leiab *osakonna_nr* tabelist *Osakond* ja koostab ning kontrollib väiteid.
1 NOT IN (1, 2, 1, 3, NULL) => FALSE
2 NOT IN (1, 2, 1, 3, NULL) => FALSE
3 NOT IN (1, 2, 1, 3, NULL) => FALSE
4 NOT IN (1, 2, 1, 3, **NULL**) => **UNKNOWN**

Viimase väite kontrollimise tulemus on UNKNOWN, sest võibolla on teadmata element (NULL) 4, aga võibolla mitte. Päringu tulemuses on andmed ainult selliste osakondade kohta, mille korral konstrueeritakse otsingutingimuse täidetuse kontrollimiseks väide, mis on TRUE. Seega poleks antud juhul päringu tulemuses andmeid mitte ühegi osakonna kohta. Seega on antud juhul alampäringus kasutusel tingimus *osakonna_nr IS NOT NULL*.

Näide: Leida iga töötajaid omava osakonna kõige väiksema palga saaja. Lahendus MS Accessis:

```
SELECT *
FROM Tootaja
WHERE (palk & ' ' & osakonna_nr) IN (SELECT Min(palk) & ' ' & osakonna_nr
AS combo FROM Tootaja GROUP BY osakonna_nr)
AND osakonna_nr IS NOT NULL;
```

Kui predikaadis sisalduv alampäring võib tagastada mitu rida, siis ei saa kasutada võrdluse predikaati, vaid tuleb kasutada IN predikaati.

5.3 EXISTS predikaat

```
<exists predikaat>::=[NOT] EXISTS <tabeli alampäring>
```

EXISTS tingimus on täidetud (TRUE) siis ja ainult siis, kui alampäring leiab vähemalt ühe rea. EXISTS tingimus on täitmata (FALSE) siis, kui alampäring ei leia ühtegi rida.

Näide: Kui andmebaasis on registreeritud vähemalt üks töötaja, siis leia kõikide osakondade andmed.

```
SELECT *
FROM Osakond
WHERE EXISTS (SELECT * FROM Tootaja);
```

Selles lauses kasutatakse mittekorreleeruvat alampäringut.
Näide: Leia kõik osakonnad, kus on vähemalt üks töötaja.

```
SELECT *
FROM Osakond
WHERE EXISTS (SELECT *
FROM Tootaja
WHERE Tootaja.osakonna_nr=Osakond.osakonna_nr);
```

Antud lahendus kasutab *korreleeruvat* alampäringut (enamasti kasutataksegi EXISTS predikaati koos korreleeruva alampäringuga). Alampäringu poole pöördutakse korduvalt – iga tabeli *Osakond* rea puhul. Alampäringuga leitakse konkreetselt selles osakonnas töötavad töötajad. Kui alampäring leiab kasvõi ühegi rea, siis on tingimus täidetud ja vaadeldav osakond võetakse põhipäringu tulemusse. WHERE klausel on oluline, et vaadelda iga osakonna juures vaid seal töötavaid töötajaid.

Kui WHERE klausel alampäringust ära jätta siis oleks päringu tulemuseks:

- Kui tabelis *Tootaja* on kasvõi ühe töötaja andmed, siis leiab alampäring rea, tingimus on täidetud ja kasutajale näidatakse **kõikide osakondade andmeid**.

```
SELECT *
FROM Osakond
WHERE TRUE;
```

See on ekvivalentne (samaväärne) järgneva päringuga:

```
SELECT *
FROM Osakond;
```

- Kui tabelis *Tootaja* pole ühtegi rida, siis on tingimus täitmata ja tulemusena ***ei väljasta ühtegi rida***.

Leidmaks kõiki osakondi, kus pole ühtegi töötajat, võib teha päringu:

```
SELECT *
FROM Osakond
WHERE NOT EXISTS (SELECT * FROM Tootaja AS T WHERE
Tootaja.osakonna_nr=Osakond.osakonna_nr);
```

Antud päringus olev tingimus on täidetud, kui alampäring ei leia ühtegi rida. Tegemist on päringuga, mis kasutab korreleeruvat alampäringut. Mäletatavasti sai selle ülesande lahendada ka mittekorreleeruva alampäringuga:

```
SELECT *
FROM Osakond
WHERE osakonna_nr NOT IN (SELECT osakonna_nr
        FROM Tootaja
        WHERE osakonna_nr IS NOT NULL);
```

Andmebaasisüsteem *võib* (aga ei pruugi) olla piisavalt arukas (tegelikult määravad selle "arukuse" andmebaasisüsteemi loojad), mõistmaks, et eelnevalt esitatud kaks SQL lauset (korreleeruva ja mittekorreleeruva alampäringuga) annavad sama tulemuse ja neile võib koostada sama ***täitmisplaani***.

Pange tähele, et EXISTS predikaadi kasutamise ei pea alampäringu SELECT klauslis nimetama konkreetseid veerge. Antud päringute puhul ei võrrelda veergudes olevaid väärtuseid vaid kontrollitakse, kas alampäring leiab mõne rea või mitte.

IN ja EXISTS predikaate, mis sisaldavad alampäringuid, saab kasutada relatsioonialgebra operatsiooni "lõige" realiseerimiseks.

IN ja EXISTS predikaate, mis sisaldavad alampäringuid ja eitust, saab kasutada relatsioonialgebra operatsiooni "vahe" realiseerimiseks.

5.4 Koguselised võrdluse predikaadid

```
<koguseline predikaat>::=<väärtuse avaldis> <võrdlusoperaator> {ANY |
SOME | ALL} <alampäring>
```

```
<võrdlusoperaator>::={= | <> | < | > | <= | >=}
```

Võimaldab võrrelda väärtust mingisse väärtuste hulka kuuluvate väärtustega. Väärtuste hulk tuleb leida alampäringuga.

5.4.1 ANY e SOME

Tingimus loetakse täidetuks, kui *ühe või rohkema* väärtuste hulka kuuluva väärtuse korral on võrdluse tulemus TRUE. Väärtuste hulk tuleb leida alampäringuga. Sõnad ANY ja SOME on sünonüümid. Kui alampäringu abil leitud tabelis on 0 rida, siis tingimus ei ole täidetud.

<u>Näide</u>: Leia töötajad, kelle palk on suurem vähemalt ühest osakonna nr 1 töötaja palgast ja kes ise ei tööta osakonnas nr 1.

```
SELECT *
FROM Tootaja
WHERE palk > ANY (SELECT palk FROM Tootaja WHERE osakonna_nr=1)
AND (osakonna_nr<>1 OR osakonna_nr IS NULL);
```

See lahendus kasutab mittekorreleeruvat alampäringut.
Predikaati "palk > SOME (SELECT palk FROM Tootaja WHERE osakonna_nr=1)" kasutatakse, et kontrollida, kas töötaja palk on suurem kui vähemalt ühel osakonnas nr 1 töötaval isikul. Alampäringuga leitakse palkade hulk. Põhipäringus võrreldakse põhipäringus leitud töötajate palku alampäringu poolt leitud palkade hulka kuuluvate palkadega. Kui töötaja *t* palk on suurem kui vähemalt üks (ANY) alampäringu poolt leitud palk ja kehtib tingimus *(t.osakonna_nr<>1 OR t.osakonna_nr IS NULL)*, siis kuuluvad *t* andmed päringu tulemusse.

See päring on samaväärne kui:

```
SELECT *
FROM Tootaja
WHERE palk > SOME (SELECT palk FROM Tootaja WHERE
osakonna_nr=1) AND (osakonna_nr<>1 OR osakonna_nr IS NULL);
```

või:

```
SELECT *
FROM Tootaja
WHERE palk > (SELECT Min(palk) FROM Tootaja WHERE osakonna_nr=1)
AND (osakonna_nr<>1 OR osakonna_nr IS NULL);
```

Viimases päringus kontrollitakse, et töötaja palk oleks suurem kui minimaalne palk, mida makstakse osakonnas nr 1. Teises alamtingimuses kontrollitakse, et töötaja ei töötaks osakonnas nr 1.

<u>Näide</u>: Leia osakonnad, kus töötab vähemalt üks töötaja.

Järgnevad päringud on samaväärsed.

```
SELECT *
FROM Osakond
WHERE osakonna_nr IN (SELECT osakonna_nr FROM Tootaja
WHERE osakonna_nr IS NOT NULL);
```

```
SELECT *
FROM Osakond
WHERE osakonna_nr = ANY (SELECT osakonna_nr FROM Tootaja
WHERE osakonna_nr IS NOT NULL);
```

NB! Järgnev päring ei tööta, kui alampäring leiab rohkem kui ühe rea.

```
SELECT *
FROM Osakond
WHERE osakonna_nr = (SELECT osakonna_nr FROM Tootaja
WHERE osakonna_nr IS NOT NULL);
```

Võrdsuse kontrolli operaatorit (=) saab kasutada sellisel viisil vaid juhul, kui alampäring on skalaarne alampäring, mille tulemuseks on skalaarne väärtus.

5.4.2 ALL

Tingimus loetakse täidetuks, kui *kõigi* väärtuste hulka kuuluvate väärtuste korral on võrdluse tulemus TRUE. Väärtuste hulk tuleb leida alampäringuga. Tingimus loetakse täidetuks, kui alampäringu abil leitud tabelis on 0 rida.

Näide: Leia töötajad, kelle palk on kõrgem kõigi osakonna nr 1 töötajate palkadest ja kes ise ei tööta osakonnas nr 1.

```
SELECT *
FROM Tootaja
WHERE palk > ALL (SELECT palk FROM Tootaja WHERE osakonna_nr=1)
AND (osakonna_nr<>1 OR osakonna_nr IS NULL);
```

Alampäringuga leitakse palkade hulk. Põhipäringus võrreldakse põhipäringus leitud töötajate palku alampäringu poolt leitud palkade hulka kuuluvate palkadega. Kui töötaja *t* palk on suurem kui kõik (ALL) alampäringu poolt leitud palgad ja kehtib tingimus *(t.osakonna_nr<>1 OR t.osakonna_nr IS NULL)*, siis kuuluvad *t* andmed päringu tulemusse.

See päring on samaväärne kui:

```
SELECT *
FROM Tootaja
WHERE palk > (SELECT Nz(Max(palk),0) AS maks FROM Tootaja WHERE
osakonna_nr=1) AND (osakonna_nr<>1 OR osakonna_nr IS NULL);
```

MS Accessi süsteemi-defineeritud funktsioon *Nz*, mis asendab NULLi mingi väärtusega (antud juhul 0), on vajalik olukorras, kui alampäringu tingimusele ei vasta mitte ükski rida ning Max funktsioon tagastab NULL.

Päring, kus on tingimus *> ALL* tagastab olukorras, kui alampäringu tingimusele ei vasta mitte ükski rida, kõik read tabelist *Tootaja. Samas* ilma funktsiooni *Nz* kasutamiseta ei oleks alumise päringu tulemuses mitte ühtegi rida.

Date (2009) märgib, et koguselist võrdluse predikaat kasutav tingimus on enamasti võimalik asendada loogiliselt samaväärse tingimusega, kus taolist predikaati ei kasutata. Vajadusel tuleb kasutada *Nz* (*Coalesce*) funktsiooni. Date (2009) esitab teisenduste kohta järgneva tabeli.

	ANY	ALL
=	IN	
<>		NOT IN
<	< MAX	< MIN
<=	<= MAX	<= MIN
>	> MIN	> MAX
>=	>= MIN	>= MAX

5.5 Unique predikaat

Võimaldab kontrollida, kas alampäringu poolt leitavate ridade hulk sisaldab korduvaid ridu või on kõik read seal väärtuse poolest unikaalsed.

```
<unique predikaat>::= [NOT] UNIQUE <tabeli alampäring>
```

Tagastab TRUE, kui alampäring leiab 0 rida, 1 rida või kui selle leitud ridade hulgas ei sisaldu korduvaid ridu (read on unikaalsed). Vastasel juhul tagastab FALSE.

Näide: Väljasta kõigi osakondade kõik andmed ainult siis, kui iga nimi esineb andmebaasis ainult üks kord.

```
SELECT *
FROM Osakond
WHERE UNIQUE (SELECT osakonna_nimi FROM Osakond);
```

5.6 Match predikaat

Võimaldab kontrollida, kas reale leidub osaline või täielik vaste alampäringu poolt leitavate ridade hulgas.

```
<match predikaat>::= <rea väärtuse konstruktor> MATCH [UNIQUE] [SIMPLE
| PARTIAL | FULL] <tabeli alampäring>
```

<rea väärtuse konstruktor> ja <tabeli alampäring> poolt leitavad read peavad sisaldama sama arvu välju ning vastavate väljade paarid peavad olema võrreldavate andmetüüpidega.

Tagastab TRUE, kui <rea väärtuse konstruktor> poolt esitatud reale leidub vaste alampäringu poolt leitud ridade seas.

5.7 Rohkem kui ühte alampäringut sisaldavad päringud

Ingl *compound nested queries*. Päringud võivad sisaldada rohkem kui ühte alampäringut. Mitu alampäringut sisaldavat alamtingimust võivad paikneda samal tasemel ja olla ühendatud loogikaoperatsioonide AND või OR kaudu.

Näide: Leia osakonnad (kõik andmed), kus töötavad töötajad nr 1 ja nr 2.

```
SELECT *
FROM Osakond
WHERE (osakonna_nr IN
(SELECT osakonna_nr FROM TOOTAJA WHERE tootaja_kood=1)
OR
osakonna_nr IN
(SELECT osakonna_nr FROM TOOTAJA WHERE tootaja_kood=2));
```

Nagu SQLis tavaline, saab ülesande lahendada mitmel viisil. Järgmises lahenduses kasutatakse alampäringu asemel joini.

```
SELECT DISTINCT Osakond.osakonna_nr, Osakond.osakonna_nimi
FROM Osakond INNER JOIN Tootaja ON
Osakond.osakonna_nr=Tootaja.osakonna_nr
WHERE tootaja_kood=1 OR tootaja_kood=2;
```

DISTINCT läheb praegu vaja selleks, et juhul, kui töötajad koodiga 1 ja 2 töötavad mõlemad samas osakonnas, ei väljastataks selle osakonna andmeid mitmekordselt. SQLis ei eemaldata korduvaid ridu automaatselt, vaid selleks tuleb eraldi käsk anda (v.a UNION, INTERSECT, EXCEPT/MINUS operatsioonide tulemus).

Näide: Leia töötajad, kes saavad keskmisest rohkem palka, aga ei saa kõige suuremat palka.

```
SELECT *
FROM Tootaja
WHERE palk>(SELECT Avg(palk) AS keskmine FROM Tootaja)
AND palk<>(SELECT Max(palk) AS maksimaalne FROM Tootaja);
```

Alampäringud võivad olla ka teiste alampäringute sees (nende WHERE klauslites) st siis mitte enam samal tasemel. Ajalooliselt on välja kujunenud, et andmebaasisüsteemides on sageli lubatud maksimaalne tasemete arv 16. Samas erinevates süsteemides on see arv erinev. Näiteks MS Access 2019 puhul on lubatud maksimaalne tasemete arv 50 (https://support.office.com/en-us/article/Access-2016-specifications-0cf3c66f-9cf2-4e32-9568-98c1025bb47c). Alampäringute täitmise järjekord on seestpoolt väljapoole.

Näide: Leia osakonnad, kus töötab töötajaid, kes saavad keskmisest rohkem kui 200 eurot rohkem palka.

```
SELECT *
FROM Osakond
WHERE osakonna_nr IN
	(SELECT osakonna_nr FROM Tootaja WHERE palk>
		(SELECT Avg(palk) FROM Tootaja) + 200);
```

Alampäring võib asetseda mõlemal pool võrdlusoperaatorit.

```
SELECT *
FROM Osakond
WHERE osakonna_nr IN
(SELECT osakonna_nr FROM Tootaja WHERE
 ((SELECT Avg(palk) FROM Tootaja) + 200)<palk);
```

```
SELECT *
FROM Osakond
WHERE osakonna_nr IN
(SELECT osakonna_nr FROM Tootaja WHERE
 (SELECT Avg(palk) FROM Tootaja) <palk-200);
```

Näide: Leia osakond, milles on rohkem alla 1500 euro palka saavaid töötajaid kui üle 1800 euro palka saavaid töötajaid.

```
SELECT O.osakonna_nr, O.osakonna_nimi
FROM Osakond O
WHERE
	(SELECT Count(*)
	FROM Tootaja T
	WHERE T.osakonna_nr=O.osakonna_nr AND T.palk<1500)>
		(SELECT Count(*)
		FROM Tootaja T
		WHERE T.osakonna_nr=O.osakonna_nr AND T.palk>1800);
```

Nagu selles päringus näete võib aliaste määramisel ka reserveeritud sõna AS mitte kasutada.

5.8 Alampäringud mujal kui WHERE klauslis

Alampäringud võivad SELECT lauses sisalduda ka SELECT, FROM ja HAVING klauslis.

Näide: Leia iga osakonna töötajate arv.

```
SELECT osakonna_nimi, (SELECT Count(*) FROM Tootaja AS T WHERE
T.osakonna_nr=O.osakonna_nr) AS arv
FROM Osakond AS O;
```

NB! Date (2009) märgib, et järgnevad kaks päringut ei ole loogiliselt samaväärsed.

a)
```
SELECT osakonna_nimi, (SELECT Count(*) FROM Tootaja AS T WHERE
T.osakonna_nr=O.osakonna_nr) AS arv
FROM Osakond AS O;
```

b)
```
SELECT osakonna_nimi, (SELECT Count(*) AS arv FROM Tootaja AS T
WHERE T.osakonna_nr=O.osakonna_nr)
FROM Osakond AS O;
```

Päringu a) tulemuseks olevas tabelis on veerud *osakonna_nimi* ja *arv.* Päringu b) tulemuseks olevas tabelis on veerg *osakonna_nimi*. Kuid veerul, milles olevad andmed leitakse alampäringu abil, on realisatsioonist sõltuv nimi.

Näide: Leia üle 1500 euro palka saavad töötajad.
```
SELECT tootaja_kood, osakonna_nr, palk
FROM (SELECT * FROM Osakond
        WHERE palk>1500) AS suurem_palk;
```
suurem_palk – alampäringuga moodustatud virtuaalse tabeli nimi (alias). See on päringu koostaja poolt välja mõeldud. Selle nime määramine on keelereeglite poolt nõutud.

Näide: Leia sellised töötaid omavad osakonnad, kus on töötajaid rohkem kui töötajaid omavates osakondades keskmiselt.

```
SELECT Osakond.osakonna_nr, Osakond.osakonna_nimi,
Count(Tootaja.tootaja_kood) AS arv
FROM Osakond INNER JOIN Tootaja ON Osakond.osakonna_nr =
Tootaja.osakonna_nr
GROUP BY Osakond.osakonna_nr, Osakond.osakonna_nimi
HAVING Count(Tootaja.tootaja_kood) > (SELECT Avg(arv) FROM (SELECT
Count(*) AS arv FROM Tootaja GROUP BY osakonna_nr));
```

5.9 Piirangud alampäringutele

SQL:1999 standard seab alampäringule järgmised piirangud (Gulutzan ja Pelzer, 1999).

- Alampäring ei tohi sisaldada ORDER BY klauslit.
- Alampäring ei tohi olla kokkuvõttefunktsiooni argumendiks ...Avg((SELECT veerg1 FROM Tabel1 ...)). Näib, et hilisemates standardi versioonides on seda piirangut lõdvendatud ning kokkuvõttefunktsiooni argumendiks võib olla *skalaarne alampäring*.
- Alampäringu SELECT klauslis ei tohi olla BLOB, CLOB, NLOB või ARRAY tüüpi veerge.

5.10 Tabelite võrdlemine

Relatsiooniline mudel näeb ette, et peab leiduma võrdlusoperaator, mis võimaldab omavahel võrrelda kahte relatsiooni. SQLis paraku sellist operaatorit ei leidu. Selleks, et võrrelda omavahel kas kaks tabelit on võrdsed või mitte võib kasutada päringut, milles kasutatakse alampäringuid. Olgu meil tabelid Linn1 ja Linn2, milles on järgnevad andmed.

Linn1

kood	nimi
0784	Tallinn
0795	Tartu linn
0625	Pärnu linn
0349	Kuressaare linn

Linn2

kood	nimi
0795	Tartu linn
0183	Haapsalu linn
0566	Paide linn
0919	Võru linn

Joonis 3 Näites kasutatavad tabelid.

Mõlemas tabelis on primaarvõti (kood) ja alternatiivõti (nimi).

Olgu meie ülesandeks leida kas tabelis *Linn1* ja *Linn2* on täpselt samasugused linnade koodid. Ülesande lahendamiseks võib kirjutada päringu (Date, 2009):

```
SELECT 'TRUE' AS vastus
FROM Dual
WHERE NOT EXISTS (SELECT kood FROM Linn1 WHERE NOT EXISTS
(SELECT kood FROM Linn2 WHERE Linn1.kood=Linn2.kood))
AND NOT EXISTS (SELECT kood FROM Linn2 WHERE NOT EXISTS
(SELECT kood FROM Linn1 WHERE Linn1.kood=Linn2.kood));
```

Päringu tulemuseks on tabel, kus on üks veerg ja null või üks rida. Tabelis on üks rida kus on tekstitüüpi väärtus "TRUE", kui tabelis *Linn1* ja *Linn2* on täpselt samasugused koodid. Tabelis ei ole ühtegi rida, kui tabelis *Linn1* ja *Linn2* ei ole täpselt ühesugused koodid.

Dual on spetsiaalselt selle ülesande lahendamiseks loodud abitabel, kus on üks veerg ja üks rida.

Päringus kasutatava tingimuse võib kirjutada lahti järgnevalt.
- Ei tohi leiduda ühtegi koodi tabelis *Linn1*, millele ei leidu vastavat koodi tabelis *Linn2* ja
- ei tohi leiduda ühtegi koodi tabelis *Linn2*, millele ei leidu vastavat koodi tabelis *Linn1*.

6. Grupeerimine

Grupeerimist võib ka nimetada rühmitamiseks, sest see võimaldab ridu rühmadeks e gruppideks e hulkadeks kokku võtta ja iga rühma (grupi, hulga) kohta arvutada kokkuvõttefunktsiooni kasutades koondväärtuse. Kokkuvõttefunktsioonid annavad väärtuse ridade grupi kohta. ***SQL:1992 standardi poolt kirjeldatavad kokkuvõttefunktsioonid)***.

- Min – minimaalse väärtuse leidmine.
- Max – maksimaalse väärtuse leidmine.
- Sum – summaarse väärtuse leidmine.
- Avg – keskmise väärtuse leidmine.
- Count – eksemplaride arvu leidmine.

Olgu tegemist laosüsteemiga, kus kaup saabub lattu partiidena. Kauba sisse tulemisel registreeritakse kauba tüüp, kogus ja kauba lattu tulemise aeg.

Tabel: Ladu

sissetuleku_kood	kauba_tüüp	kogus	sissetuleku_aeg
1	Kartul	1000	1. september 2000
2	Kartul	150	1. september 2000
3	Kapsas	2000	10. september 2000
4	Kaalikas	100	4. september 2000
5	Kapsas	215	3. august 2000
6	Kartul	200	2. september 2000
7	Seller		

Kui SQL lauses ei kasutata GROUP BY klauslit, siis loetakse päringu tulemusel leitud read üheks grupiks. Kui SQL lauses kasutatakse GROUP BY klauslit, siis moodustatakse grupid GROUP BY klauslis näidatud veergude unikaalsete väärtuste kombinatsioonide põhjal.

Päringus:

```
SELECT Count(*) AS arv
FROM Ladu;
```

moodustavad kõik tabeli read ühe grupi. Grupeerida võib ühes või mitmes veerus olevate väärtuste põhjal:

Olgu ülesandeks leida erinevate kaupade tüüpide koguhulk laos (laoseis):

Kartul = 1000 + 150 +200 = 1350 kg,
Kapsas = 2000 + 215 = 2215 kg,
Kaalikas = 100 kg,
Seller – määramata, kuna puuduvad andmed.

Sama tulemuse annab SQL päring:

```
SELECT kauba_tüüp, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp;
```

kauba_tüüp	koguhulk
Kartul	1350
Kapsas	2215
Kaalikas	100
Seller	

Lause osa "GROUP BY kauba_tüüp" näitab, et grupid, mille sees kauba kogused *Sum* funktsiooni abil summeeritakse, luuakse veerus *kauba_tüüp* toodud väärtuste abil.

Selliseid päringuid kasutatakse statistika tegemiseks. Pange tähele, et kõik veerud, mis on esitatud SELECT klauslis ja millele ei ole rakendatud kokkuvõttefunktsioone, peavad olema esitatud GROUP BY klauslis.

Seega päring:

```
SELECT kauba_tüüp, sissetuleku_aeg, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp;
```

annab veateate, sest GROUP BY klauslis pole grupeeritud sissetuleku aja järgi.

Miks andmebaasisüsteem seda päringut ei täida? Ühte tüüpi kaupa võis lattu tulla erinevatel aegadel. Milline väärtus esitada päringu tulemuses veerus *sissetuleku_aeg*? Võimalikud vastused oleksid näiteks "Mistahes selle kauba tüübi sissetuleku aeg" või "Kõige viimane sellise kauba tüübi sissetuleku aeg", kuid sellisel viisil päringu täitmine kompenseeriks tegelikult kasutaja poolt tehtud sisulist viga.

Küll aga töötaks päring:

```
SELECT kauba_tüüp, sissetuleku_aeg, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg
ORDER BY kauba_tüüp, sissetuleku_aeg;
```

kauba_tüüp	sissetuleku_aeg	koguhulk
Kaalikas	4. september 2000	100
Kapsas	3. august 2000	215
Kapsas	10. september 2000	2000
Kartul	1. september 2000	1150
Kartul	2. september 2000	200
Seller		

1. septembril 2000 tuli kartulit lattu 2 partiid. Nende kaal on summeeritud, kuna grupeerimistingimuses oli ka sissetuleku aeg

Samuti töötaksid järgnevad päringud, ainult nende väljastatav tulemus ei ole kasutajatele ilmselt just kuigi kasulik ja paljuütlev:

```
SELECT kauba_tüüp, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg
ORDER BY kauba_tüüp, sissetuleku_aeg;
```

kauba_tüüp	koguhulk
Kaalikas	100
Kapsas	215
Kapsas	2000
Kartul	1150
Kartul	200
Seller	

```
SELECT Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg;
```

koguhulk
100
215
2000
1150
200

Kui SELECT lause sisaldab GROUP BY klauslit, siis SELECT klausel võib sisaldada:

- konstandid;
- kokkuvõttefunktsioonide väljakutsed;
- veerud, mis on määratud GROUP BY klauslis.

MS Accessi SQLi mägimurrak nõuab, et igal juhul peavad GROUP BY klauslis olema sellised veeru nimed, mis osalevad SELECT klauslis, ning millele ei ole rakendatud kokkuvõttefunktsioone. SQL standard annab siinkohal veidi suuremaid vabadusi ja lubab jätta GROUP BY klauslis viitamata SELECT klauslis nimetatud veerule (millele pole rakendatud kokkuvõttefunktsiooni), juhul kui andmebaasisüsteemil on võimalik andmebaasis deklareeritud võtmete alusel aru saada, et selles veerus on kõigi gruppi kuuluvate ridade korral alati ühesugune väärtus.

GROUP BY ja ORDER BY klauslis võib olla lisaks selliseid veeru nimesid, mida SELECT klauslis ei ole.

Mõelge operatsioonide loogilisele järjekorrale (vt teema 3) – GROUP BY viiakse läbi enne SELECT'i. SELECT ei saa valida veerge, mida GROUP BY operatsiooni tulemuses ei ole.

Grupeerimise juures arvestage kindlasti sellega, et valiksite GROUP BY klauslisse sellised veerud, mis iga grupi üheselt identifitseerivad. Näiteks kui teete päringu inimeste kohta ja grupeerite ainult nende perekonnanime järgi, siis on tulemus suure tõenäosusega ebakorrektne, sest andmebaasis võib

olla mitu ühesuguse perenimega isikut, kes nüüd üheks grupiks kokku loetakse. Seega GROUP BY klauslis peaks üldjuhul sisalduma kandidaatvõti.

HAVING piirab nende GROUP BY järgi grupeeritud gruppide väljastamist, mis ei vasta otsingutingimusele. Seega võimaldab HAVING klausli kasutamine piirata väljastatavaid ridu kokkuvõttefunktsiooni tulemuse alusel.

Näide: Leia kauba sissetulek päevade kaupa. Väljasta vaid sellised read, kus päevane sissetulnud kogus on üle 1500 ühiku.

```
SELECT kauba_tüüp, sissetuleku_aeg, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg
HAVING Sum(kogus)>1500
ORDER BY kauba_tüüp, sissetuleku_aeg;
```

kauba_tüüp	sissetuleku_aeg	koguhulk
Kapsas	10. september 2000	2000

Selle ülesande saab lahendada ka ilma HAVING klauslita:

```
SELECT *
FROM (SELECT kauba_tüüp, sissetuleku_aeg, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg) AS ap
WHERE koguhulk>1500
ORDER BY kauba_tüüp, sissetuleku_aeg;
```

Päring tehakse virtuaalse tabeli põhjal, mis moodustatakse alampäringut kasutades.

Selle sama ülesande lahendamiseks ilma GROUP BY klauslita võib kasutada järgmist päringut:

```
SELECT DISTINCT kauba_tüüp, sissetuleku_aeg, (SELECT Sum(kogus)
FROM Ladu AS La WHERE L.kauba_tüüp=La.kauba_tüüp AND
L.sissetuleku_aeg=La.sissetuleku_aeg) AS koguhulk
FROM Ladu AS L
WHERE (SELECT Sum(kogus) FROM Ladu AS La WHERE
L.kauba_tüüp=La.kauba_tüüp AND L.sissetuleku_aeg=La.sissetuleku_aeg)
> 1500
ORDER BY kauba_tüüp, sissetuleku_aeg;
```

Date (2009) märgib, et GROUP BY ja HAVING klauslid on muutunud loogiliselt üleliigseteks – iga relatsiooniline avaldis, milles neid kasutatakse, on võimalik ümber kirjutada nii, et neid klausleid ei kasutata. Kuid erinevalt UNION JOINist kasutatakse selliseid lausekonstruktsioone laialdaselt ning seetõttu ei saa neid ka keelest eemaldada.

SQL lubab kasutada ORDER BY klauslis arvutuse tulemusena moodustatud veeru aliast. MS Accessis see ei toimi. Järgnev päring MS Accessis ei töötaks.

```
SELECT kauba_tüüp, sissetuleku_aeg, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg
HAVING koguhulk>1500
ORDER BY koguhulk;
```

Küll aga töötaks päring:

```
SELECT kauba_tüüp, sissetuleku_aeg, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg
HAVING Sum(kogus)>1500
ORDER BY Sum(kogus);
```

HAVING klauslit võib põhimõtteliselt kasutada ka ilma GROUP BY klauslita, kuid seda tehakse harva. Järgnev päring annab tulemuseks tabelis *Ladu* olevate ridade arvu, kui tabelis on üle 5 rea. Kui tabelis on 5 rida või vähem, siis päring tulemust ei tagasta.

```
SELECT Count(*) AS arv
FROM Ladu
HAVING Count(*)>5;
```

Näide: Leia, kui mitu korda on igat tüüpi kaupa lattu sisse tulnud.

```
SELECT kauba_tüüp, Count(*) AS sissetulekute_arv
FROM Ladu
GROUP BY kauba_tüüp;
```

Näide: Leia, kui mitu erinevat tüüpi kaupa on lattu tulnud.

```
SELECT Count(DISTINCT kauba_tüüp) AS arv
FROM Ladu;
```

MS Accessis (2096) paraku selline päring ei tööta ja ülesande lahendab päring, mis on samuti SQL mõttes täiesti legaalne:

```
SELECT Count(*) AS arv
FROM (SELECT DISTINCT kauba_tüüp FROM Ladu) AS t;
```

Näide: Leia kauba tüüpide kaupa sissetulekute arv, mis on olnud väiksema kogusega kui 1500. Väljasta kauba tüüp ja sissetulekute arv. Tulemuses peavad olema vaid kauba tüübid, millel on vähemalt üks selline sissetulek.

```
SELECT kauba_tüüp, Count(*) AS sissetulekute_arv
FROM Ladu
WHERE kogus<1500
GROUP BY kauba_tüüp;
```

WHERE klauslis oleva tingimusega saab piirata ridu, mida hakatakse grupeerima.

<u>Näide</u>: Leia, milline on keskmine lattu saabunud kauba tüübi partii suurus

```
SELECT kauba_tüüp, Avg(kogus) AS keskmine_kogus
FROM Ladu
GROUP BY kauba_tüüp;
```

<u>Näide</u>: Leiab, milline on iga kauba tüübi puhul kõige hilisem aeg, millal seda on lattu toodud.

```
SELECT kauba_tüüp, Max(sissetuleku_aeg) AS viimane_sissetulek
FROM Ladu
GROUP BY kauba_tüüp;
```

<u>Näide</u>: Tee tabeli *Ladu* põhjal statistikat kauba tüüpide kohta.

```
SELECT kauba_tüüp, Count(*) AS sissetulekute_arv, Avg(kogus) AS
keskmine_kogus, Min(kogus) AS min_kogus, Max(kogus) AS max_kogus
FROM Ladu
GROUP BY kauba_tüüp;
```

<u>Näide</u>: Leia selliste kaupade tüübid, mille puhul kõik sissetulekud on tulnud ühel ja samal päeval.

```
SELECT kauba_tüüp
FROM Ladu
GROUP BY kauba_tüüp
HAVING Min(sissetuleku_aeg)=Max(sissetuleku_aeg);
```

Selliste kauba tüüpide puhul on minimaalne ja maksimaalne sissetuleku päev ühesugune. *Lahenduse eeldus* – sissetuleku aeg on registreeritud kuupäeva täpsusega.

<u>Näide</u>: Leia iga osakonna kohta, kui mitu töötajat seal töötab. Päringu tulemuses peab olema osakonna number, nimi ja töötajate arv.

```
SELECT Osakond.osakonna_nr, Osakond.osakonna_nimi,
Count(Tootaja.tootaja_kood) AS arv
FROM Osakond LEFT JOIN Tootaja ON Osakond.osakonna_nr =
Tootaja.osakonna_nr
GROUP BY Osakond.osakonna_nr, Osakond.osakonna_nimi;
```

GROUP BY ja JOIN võib kasutada ühes lauses. Ühendamise tulemus:

osakonna_nr	osakonna_nimi	tootaja_kood
1	personaliosakond	1
1	personaliosakond	3
2	kinnisvaraosakond	2
3	müügiosakond	4
4	lepingute osakond	

Grupeerimise tulemus:

osakonna_nr	osakonna_nimi	arv
1	personaliosakond	2
2	kinnisvaraosakond	1
3	müügiosakond	1
4	lepingute osakond	0

Näide: Leia osakonnad, kus väiksema ja suurema palga vahe on üle 100 euro. Väljasta päringu tulemuses osakonna number ja nimi.

```
SELECT Osakond.osakonna_nr, Osakond.osakonna_nimi
FROM Tootaja INNER JOIN Osakond ON
Tootaja.osakonna_nr=Osakond.osakonna_nr
GROUP BY Osakond.osakonna_nr, Osakond.osakonna_nimi
HAVING Max(Tootaja.palk)-Min(Tootaja.palk)>100;
```

Näide: Leia osakonnad, kus on kõige rohkem töötajaid. Väljasta päringu tulemuses osakonna number ja nimi.

```
SELECT Osakond.osakonna_nr, Osakond.osakonna_nimi
FROM Tootaja INNER JOIN Osakond ON
Tootaja.osakonna_nr=Osakond.osakonna_nr
GROUP BY Osakond.osakonna_nr, Osakond.osakonna_nimi
HAVING Count(*)=(SELECT Max(arv) AS maks FROM (SELECT Count(*) AS
arv FROM Tootaja GROUP BY osakonna_nr) AS tootajate_arv);
```

Alternatiivne lahendus, mille puhul pole vaja kasutada kahte üksteise sees asuvat alampäringut. HAVING klauslis olev tingimus tagab, et töötajate arv osakonnas peab olema suurem või võrdne kui kõik alampäringu abil leitud töötajate arvud erinevates osakondades. Tingimus on täidetud, kui osakonna töötajate arv on võrreldes teiste osakondadega kõige suurem.

```
SELECT Osakond.osakonna_nr, Osakond.osakonna_nimi
FROM Tootaja INNER JOIN Osakond ON
Tootaja.osakonna_nr=Osakond.osakonna_nr
GROUP BY Osakond.osakonna_nr, Osakond.osakonna_nimi
HAVING Count(*)>=ALL(SELECT Count(*) AS arv FROM Tootaja GROUP
BY osakonna_nr);
```

6.1 Kokkuvõttefunktsioon ja tühjad väljad

Tabel: Ladu

sissetuleku_kood	kauba_tüüp	kogus	sissetuleku_aeg
1	Kartul	1000	1. september 2000
2	Kartul	150	1. september 2000
3	Kapsas	2000	10. september 2000
4	Kaalikas	100	4. september 2000
5	Kapsas	215	3. august 2000
6	Kartul	200	2. september 2000
7	Seller		

```
SELECT kauba_tüüp, Min(kogus) AS min_kogus, Max(kogus) AS
max_kogus, Count(*) AS sissetulekute_arv, Avg(kogus) AS keskmine_kogus,
Sum(kogus) AS summa
FROM Ladu
GROUP BY kauba_tüüp;
```

Tulemus.

kauba_tüüp	min_kogus	max_kogus	sissetulekute_arv	keskmine_kogus	summa
Kaalikas	100	100	1	100	100
Kapsas	215	2000	2	1107,5	2215
Kartul	150	1000	3	450	1350
Seller			**1**		

Ainult ***Count*** funktsiooni rakendamise tulemuseks on arv. Teiste funktsioonide rakendamise tulemuseks on NULL (väärtuse puudumine).

Oletame, et tabelis *Ladu* ei ole ühtegi rida.

```
SELECT Count(*) AS arv, Min(kogus) AS minimaalne, Max(kogus) AS
maksimaalne, Avg(kogus) AS keskmine, Sum(kogus) AS summa FROM
Ladu;
```

Tulemus:

arv	minimaalne	maksimaalne	keskmine	summa
0				

Nagu näete, leitakse väärtus (0) ainult *Count* funktsiooni rakendamise tulemusel.

6.2 Veel kokkuvõttefunktsioone

SQL standard näeb lisaks *Count, Min, Max, Sum, Avg* ette veel kokkuvõttefunktsioone (näiteks standardhälbe ja dispersiooni arvutamiseks), kuid need on andmebaasisüsteemides vähem levinud.

Näiteks MS Access (2019) võimaldab kasutada ka kokkuvõttefunktsioone nagu:
- *StdDev* ja *StDevP* – standardhälbe leidmine,
- *Var* ja *VarP* – dispersiooni leidmine,
- *First* – leiab hulgast esimese elemendi,
- *Last* – leiab hulgast viimase elemendi.

Veel üks võimalik funktsioon on *List* (MS Access seda ei paku), mis võimaldab koostada elementide nimekirja sisaldava stringi. Näide:

```
SELECT L.kauba_tüüp, List(L.kogus) AS nimekiri
FROM Ladu L
GROUP BY L.kauba_tüüp;
```

kauba_tüüp	nimekiri
Kartul	1000, 150, 200
Kapsas	2000, 215
Kaalikas	100
Seller	

Võrdluseks, PostgreSQLis (12) saab täiendavalt kasutada järgnevaid kokkuvõttefunktsioone (nimekiri pole täielik): bool_and, bool_or, bit_and, bit_or, every, string_agg, array_agg, xmlagg.

Mõned andmebaasisüsteemid (nt PostgreSQL) võimaldavad kasutajal ise kokkuvõttefunktsioone juurde programmeerida – luua kasutaja-defineeritud kokkuvõttefunktsioone (CREATE AGGREGATE lausega).

Siit veelkord moraal, et igal andmebaasis on kasutusel oma SQLi mägimurrak ning seda andmebaasisüsteemi kasutades tuleb ennast kõigepealt selle murrakuga kurssi viia.

6.3 Uuem süntaks

SQL:1999 on SQLi lisanud täiendavaid võimalusi grupeerimise läbiviimiseks:

```
SELECT kauba_tüüp, Count(*) AS sissetulekute_arv
FROM Ladu
GROUP BY GROUPING SETS (kauba_tüüp, sissetuleku_aeg,
(kauba_tüüp, sissetuleku aeg));
```

See annab sama tulemuse kui päring:

```
SELECT kauba_tüüp, sissetuleku_aeg, Count(*) AS sissetulekute_arv
FROM Ladu
GROUP BY kauba_tüüp, sissetuleku_aeg
UNION SELECT kauba_tüüp, NULL AS sissetuleku_aeg, Count(*) AS
sissetulekute_arv
FROM Ladu
GROUP BY kauba_tüüp
UNION SELECT NULL, sissetuleku_aeg, Count(*) AS sissetulekute_arv
FROM Ladu
GROUP BY sissetuleku_aeg
```

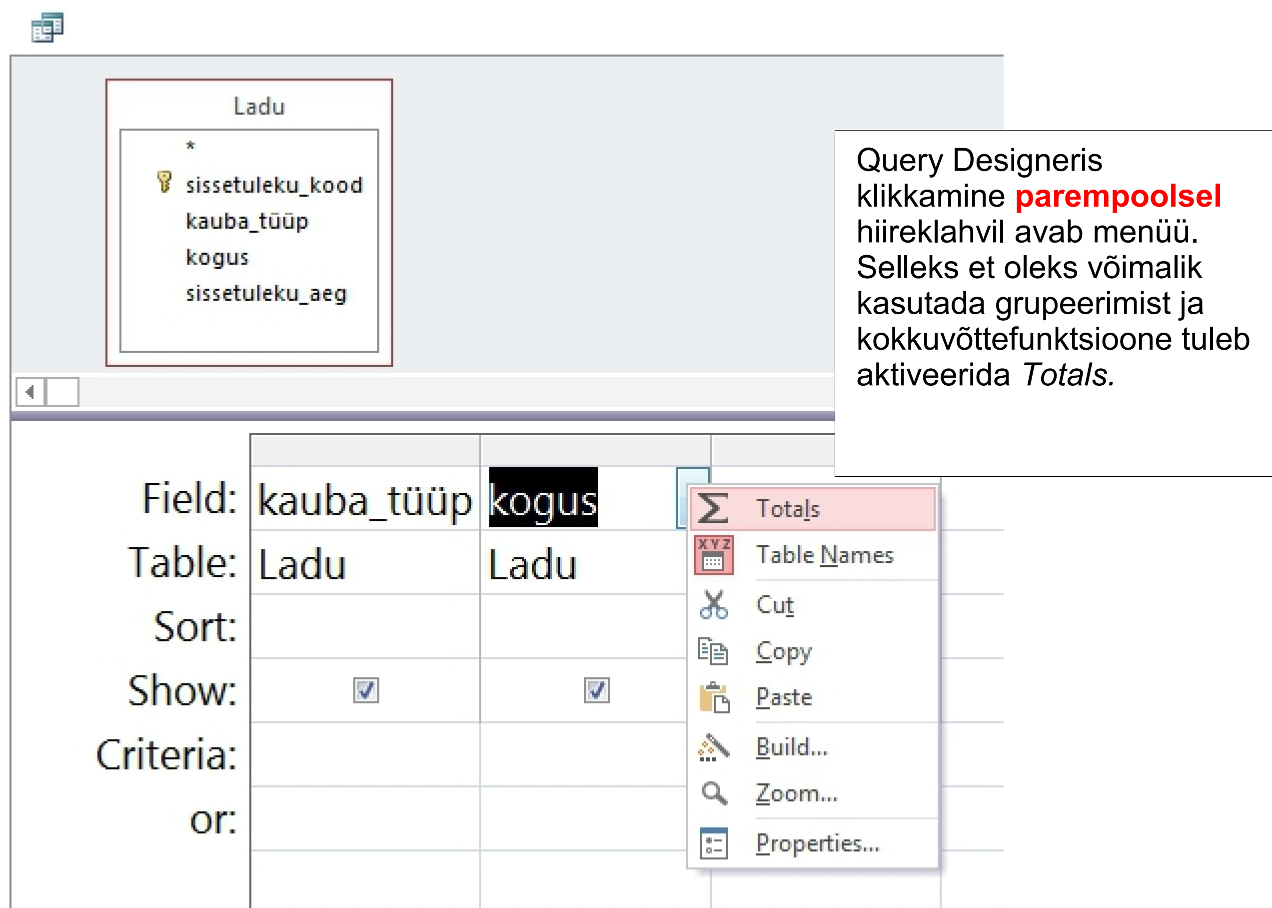
Samuti saab kasutada spetsifikatsioone CUBE ja ROLLUP.

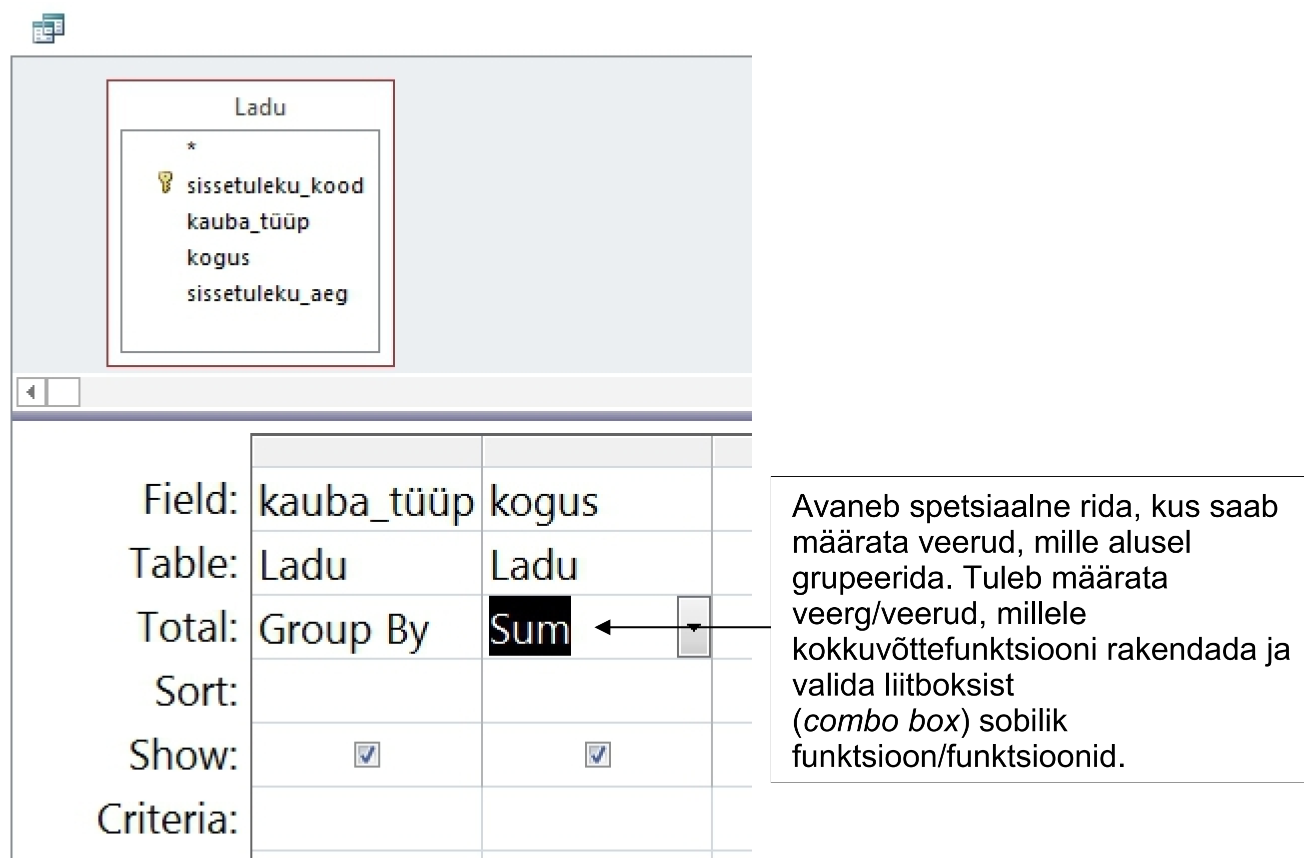
6.4 Grupeerimistingimuste lisamine ja kokkuvõttefunktsioonide kasutamine MS Accessi Query Designeris

MS Accessi *Query Designer* realiseerib tegelikult *Query by Example* visuaalse andmebaasikeele relatsiooniliste/SQL andmebaaside jaoks. Kui valite *SQL View*, siis tõlgib MS Access selles kirjutatud lause ümber SQL lauseks.

Soovin teha päringut:

```
SELECT kauba_tüüp, Sum(kogus) AS koguhulk
FROM Ladu
GROUP BY kauba_tüüp;
```


Ladu
*
sissetuleku_kood
kauba_tüüp
kogus
sissetuleku_aeg
Field: kauba_tüüp kogus
Table: Ladu Ladu
Sort:
Show:
Criteria:
or:
Totals
Table Names
Cut
Copy
Paste
Build...
Zoom...
Properties...
Query Designeris klikkamine parempoolsel hiireklahvil avab menüü. Selleks et oleks võimalik kasutada grupeerimist ja kokkuvõttefunktsioone tuleb aktiveerida Totals.



Joonis 4 Grupeerimist kasutava päringu koostamine MS Accessi Query designeris (Lause Query by Example visuaalses keeles).

Alumises näites küsitakse eelmisel loengul tutvustatud andmebaasi põhjal nende osakondade miinimum- ja maksimumpalku, kus miinimumpalk jääb alla 100 euro.

```
SELECT osakonna_nr, Min(palk), Max(palk)
FROM Tootaja
GROUP BY osakonna_nr
HAVING Min(palk) < 100;
```

6.5 Sümmeetriline vahe

Sümmeetrilise vahe leidmise operatsioon, mis rakendatakse tabelitele A ja B leiab sellised read, mis on tabelis A, kuid ei ole tabelis B ning sellised read, mis on tabelis B, kuid ei ole tabelis A. Tabelid A ja B peavad olema ühesuguse struktuuriga. Järgneva alampäringuid ning grupeerimist lahenduse pakkus välja Tropashko (2006). Ülesandeks on leida linnad, mille andmed on ainult kas tabelis *Linn1* või *Linn2*, aga mitte mõlemas korraga.

```
SELECT kood, nimi
FROM (SELECT kood, nimi,
Sum(iif(src=1,1,0)) AS cnt1,
Sum(iif(src=2,1,0)) AS cnt2
FROM
(SELECT kood, nimi, 1 AS src
FROM Linn1
UNION ALL SELECT kood, nimi, 2 AS src
FROM Linn2) AS linnade_ühend
GROUP BY kood, nimi) AS ap
WHERE cnt1<>cnt2;
```

Sisemise alampäringuga leiti tabelis *Linn1* ja *Linn2* olevate ridade ühend. Välise alampäringu abil tehakse kindlaks, millisest tabelist on pärit ühendisse kuuluvad read. Päringu tulemusse valitakse vaid sellised read, mille puhul rida ühes tabelis on, kuid teises ei ole (cnt1<>cnt).

7. SQL ja statistika

Toome näiteid mõningate statistiliste näitajate arvutamise kohta kasutades SQLi. Mõned andmebaasisüsteemid (nt Oracle) võimaldavad selliste ülesannete lahendamiseks kasutada *analüütilisi funktsioone*, kuid järgnevates näidetes neid ei kasutata.

7.1 Mediaan

Mingil kogumil mõõdetud tunnuse X variatsioonrida on sellel kogumil saadud tunnuse väärtuste jada, mis on järjestatud kasvavalt (mittekahanevalt), st jada ümberjärjestamisel saadud jada (Tiit, 2001). "palk" on üks töötaja tunnus. Leiame töötajate palkade mediaani.

Tootaja

tootaja_ kood	perenimi	aadress	registr_kpv	palk	osakonna_nr
1	Jõgi	Tallinn, Pikk 34	11.11.2001	1400	1
2	Mets	Paide, Roheline 7	12.04.2001	1500	2
3	Kask	Tartu, Tähe 12	10.05.2001	1600	1
4	Triik	Tartu, Kase 12-44	10.03.2001	1800	3
5	Tali	Põlva, Vase 3	10.05.2001	100	

Leiame tunnuse *palk* variatsioonirea: 100, 1400, 1500, 1600, 1800

Variatsioonrea keskpunkti nimetatakse *mediaaniks* med. Kui variatsioonreas on paaritu arv liikmeid, siis on mediaaniks liige järjekorranumbriga 0,5(n+1) (Tiit, 2001).

Näiteks variatsioonirea: 100, 1400, 1500, 1600, 1800 korral on mediaan liige järjenumbriga 0,5*(5+1)=0,5*6=3. Seega mediaan on 1500.

Kui variatsioonreas on paarisarv liikmeid, siis pole keskmist liiget ja mediaan defineeritakse kui kahe keskmise liikme poolsumma.

Näiteks variatsioonirea: 4000, 5000, 5000, 6000, 7000, 8000 korral on mediaan (5000+6000)/2=5500.

Mediaan jaotab variatsioonrea kaheks osaks: alumiseks (siia kuuluvad mediaanist väiksemad väärtused) ja ülemiseks (kuhu kuuluvad mediaanist suuremad väärtused) (Tiit, 2001).

Järgneva mediaani arvutamise algoritmi SQLi abil esitas Celcko (2000) ning selle autoriks on Chris Date. Esitan SQL laused töötaja palkade mediaani leidmiseks andmebaasisüsteemis MS Access. Lahendus kasutab mitut vahetulemust, mille võib realiseerida vaadetena. Kindlasti tuleb ka öelda, et selline lahendus töötab üsna aeglaselt.

Tuleks tagada, et uuritavas variatsioonireas oleks alati paarisarv elemente. Juhul, kui dubleerida iga variatsioonireas olevat elementi, siis mediaan jääb samaks.

```
SELECT palk
FROM Tootaja
WHERE palk IS NOT NULL
UNION ALL SELECT palk
FROM Tootaja
WHERE palk IS NOT NULL;
```

Salvestan päringu nime all *Ajutine1*.

Leitakse rea keskväärtuse(d):

```
SELECT palk
FROM Ajutine1
WHERE (SELECT Count(*) FROM Tootaja)<=
(SELECT Count(*) FROM Ajutine1 AS A1 WHERE A1.palk>=Ajutine1.palk)
AND
(SELECT Count(*) FROM Tootaja)<=
(SELECT Count(*) FROM Ajutine1 AS A2 WHERE A2.palk<=Ajutine1.palk);
```

Salvestan päringu nime all *Ajutine2.*

Tulemusest eemaldatakse korduvad väärtused:

```
SELECT DISTINCT palk
FROM Ajutine2;
```

Salvestan päringu nime all *Ajutine3.*

Leitakse mediaan:

```
SELECT Avg(palk) AS mediaan
FROM Ajutine3;
```

Molinaro (2005) pakub välja järgmise päringu mediaani arvutamiseks. See päring kasutab otsekorrutise leidmist ja mitmeid süsteemi-defineeritud funktsioone.

```
SELECT Avg(palk) AS mediaan
FROM (
SELECT T1.palk
FROM Tootaja AS T1, Tootaja AS T2
GROUP BY T1.palk
HAVING Sum(iif(T1.palk=T2.palk,1,0))>=Abs(Sum(Sgn(T1.palk-T2.palk)))) AS
ap
```

Funktsioon *Abs* tagastab argumendiks oleva täisarvu absoluutväärtuse. Funktsioon *Sgn* leiab argumendiks oleva arvu märgi.

7.2 Mood

Mood on hulgas kõige sagedamini esinev element. Ülesandeks on leida töötajate palkade mood.

```
SELECT palk
FROM Tootaja
GROUP BY palk
HAVING Count(palk)>= ALL (SELECT Count(palk) AS arv FROM Tootaja
GROUP BY palk);
```

Selle lahenduse pakub välja Molinaro (2005). Alampäringuga leitakse iga palga kohta seda saavate töötajate arv. Põhipäringus grupeeritakse read tabelist *Tootaja* palkade järgi ja leitakse selline palk, mida saavate töötajate arv on suurem või võrdne kõigist alampäringu abil leitud palgasaajate arvudest.

7.3 Keskmise arvutamine ilma suurimat ja väiksemat väärtust arvestamata

Ülesandeks on leida töötajate keskmine palk, kusjuures keskmise arvestamisel ei võeta arvesse kõige kõrgemat ja madalamat palka. Lähtutakse eeldusest, et ekstreemsed väärtused võivad päringu tulemust moonutada.

```
SELECT Avg(palk) AS keskmine
FROM Tootaja
WHERE palk<>((SELECT Min(palk) AS minim FROM Tootaja)) AND
palk<>(SELECT Max(palk) AS maxim FROM Tootaja);
```

7.4 Protsendi arvutamine tervikust

Ülesandeks on leida, kui mitu protsenti moodustab osakonnas 1 töötavate töötajate arv kõigi töötajate koguarvust.

Molinaro (2005) pakub välja järgneva lahenduse.

```
SELECT (Count(iif(osakonna_nr=1,osakonna_nr, NULL))/Count(*))*100 AS
protsent
FROM Tootaja;
```

Count(iif(osakonna_nr=1,osakonna_nr, NULL)) loetakse kokku ridade arv tabelis *Tootaja*, kus tingimus osakonna_nr=1 on täidetud.

Ülesande lahendamiseks võib kasutada ka päringut, kus osakonnas 1 töötavate töötajate arvu leidmiseks kasutatakse alampäringut.

```
SELECT ((SELECT Count(*) AS a FROM Tootaja WHERE
osakonna_nr=1)/Count(*))*100 AS protsent
FROM Tootaja;
```

Märkus – mõlema päringu täitmisel tekib nulliga jagamine, kui tabelis *Tootaja* on null rida. Kui soovite sellist olukorda vältida võib päringut täiendada:

```
SELECT ((SELECT Count(*) AS a FROM Tootaja WHERE
osakonna_nr=1)/iif(Count(*)=0,NULL,Count(*)))*100 AS protsent
FROM Tootaja;
```

iif(Count(*)=0,NULL,Count(*)) tagab, et kui tabelis *Tootaja* on null rida, siis jagajaks on NULL ja päringu tulemuseks on tabel, kus on üks veerg ja null rida.

7.5 Kumulatiivne (kumuleeritud) sagedus ja jaotustabel

Kumuleeritud sagedustabel arvutatakse sagedustabelist selle sagedusi summeerides (kumuleerides),

$$c_i = \sum_{j=1}^{i} n_j$$

kus c_i tähistab *i*-ndat kumuleeritud sagedust (Tiit, 2001).

Selle arvutamiseks SQLis peab olema loodud *klassifitseerimise tabel*, kus on määratud objekti klasside piirid. Tiit (2001) nimetab põhireeglid, millele sellised vahemike kirjeldused peaksid vastama.

- Klasside arvuks sobib ruutjuur vaatluste arvust. Väikese vaatluste arvu korral pigem ümmardada ülespoole, suure vaatluste arvu korral – allapoole.
- Kui on võimalik, valida klassipiirid nii, et klassid on võrdse pikkusega.
- Klassipiiride puhul tekib probleem, mida teha vaatlusega, mis sattub täpselt piirile. Sagedamini loetakse see järgmisesse klassi, va viimase klassi puhul, mille korral sageli loetakse mõlemad klassipiirid klassi kuuluvaiks.
- Kui mingis valdkonnas on olemas traditsioonilised klassipiirid, tuleks neid eelistada (näiteks rahvastikustatistikas kasutatakse viieaastaseid klasse, mis on *alt kinnised.*
- Otsmised klassid jäetakse siis lahtiseks, kui äärmised väärtused paiknevad kaugel.

Selliseks tabeliks on näiteks *Palgaaste.* Korrektse tulemuse saamiseks järgmises SQL päringus peaks see tabel olema täielik, st katma vahemikega kõik võimalikud palga suurused.

Tootaja

tootaja_kood	perenimi	aadress	registr_kpv	palk	osakonna_nr
1	Jõgi	Tallinn, Pikk 34	11.11.2001	1400	1
2	Mets	Paide, Roheline 7	12.04.2001	1500	2
3	Kask	Tartu, Tähe 12	10.05.2001	1600	1
4	Triik	Tartu, Kase 12-44	10.03.2001	1800	3
5	Tali	Põlva, Vase 3	10.05.2001	100	

Palk

astme_nr	vahemiku_algus	vahemiku_lopp
1	0	200
2	201	500
3	501	1000
4	1001	3000

```
SELECT A.vahemiku_algus, A.vahemiku_lopp, A.astme_nr, Count(palk) AS
astmesse_kuulujate_arv, ((SELECT Count(*) FROM Tootaja T2 WHERE
T2.palk<=A.vahemiku_lopp)*100)/(SELECT Count(palk) FROM Tootaja) AS
kumulatiivne_protsent
FROM Tootaja T RIGHT JOIN Palgaaste A ON (T.palk BETWEEN
A.vahemiku_algus and A.vahemiku_lopp)
GROUP BY A.vahemiku_algus, A.vahemiku_lopp,A.astme_nr;
```

Tulemuseks saadud tabel on kumuleeritud jaotustabel. Siit näeme, et 20 % palgasaajatest saavad palka kuni 100 eurot. Tulemuses ei arvestata töötajaid, kellel pole palk määratud.

vahemiku_algus	vahemiku_lopp	astme_nr	astmesse_kuulujate_arv	kumulatiivne_protsent
0	200	1	1	20
201	500	2	0	20
501	1000	3	0	20
1001	3000	4	4	100

Liiv et al. (2007) esitavad päringu, mis võimaldab kasutada SQLi andmeanalüüsis tuntud konformismianalüüsi läbiviimiseks. Nad esitavad üldise, aga samas keeruka päringu, mis töötab võimalikult paljudes erinevates andmebaasisüsteemides.

8. SQL SELECT lause täitmise järjekord

Celcko (1999) nimetab järjekorra, milles SQL SELECT lause täidetakse.

- FROM klausel.
- WHERE klausel.
- GROUP BY klausel.
- HAVING klausel
- SELECT klausel

9. Probleemsetest päringutest ja nende leidmisest

Karwin (2010) pühendab SQL-andmebaaside disaini antimustrite raamatus eraldi alajaotuse päringutega seotud probleemidele.

Karda teadmatust – NULLide kasutamine päringutes justkui oleks tegemist tavalise väärtusega. Sellise probleemi näiteks on päring

```
SELECT * FROM Tootaja WHERE osakonna_nr=NULL;
```

samas, kui õige oleks

```
SELECT * FROM Tootaja WHERE osakonna_nr IS NULL;
```

NULL ei ole väärtus ja ei käitu lausetes tavalise väärtusena. Kasutades näiteks tingimust =NULL või <>NULL on tingimuse kontrolli tulemus mistahes rea korral UNKNOWN ja päringu tulemuses ei ole ühtegi rida. Kui mõne andmebaasisüsteemi käitumine sellest erineb, siis see on standardit eirav erand.

Ebapiisav info grupeerimiseks – grupeerimise päringutes puudub GROUP BY klausel või puudub sellest veerge – peab nimetama kõik veerud, mida kirjeldatakse SELECT klauslis ja millele ei rakendata seal kokkuvõttefunktsiooni.

Vale.

```
SELECT osakonna_nr, Max(Tootaja.palk) AS suurim_palk, perenimi
FROM Tootaja
GROUP BY osakonna_nr;
```

Õige.

```
SELECT osakonna_nr, Max(Tootaja.palk) AS suurim_palk, perenimi
FROM Tootaja
GROUP BY osakonna_nr, perenimi;
```

Juhuslike ridade valimine kasutades selleks sorteerimist ja esimeste ridade valimist. Juhuslike väärtuste alusel sorteerimine (eriti suure tabeli puhul) on täitmise mõttes kallis operatsioon. Oletame, et ülesandeks on leida tabelist juhuslikult üks rida.

Halb.

```
SELECT TOP 1 *
FROM (SELECT *
FROM Tootaja
ORDER BY Rnd(tootaja_kood)) AS foo;
```

Parem. Leitakse juhuslik täisarv, mis on vahemikus kõige väiksem ja kõige suurem töötaja number. Juhusliku täisarvu leidmiseks kasutatakse valemit:

Math.floor(Math.random() * (max - min + 1) + min)
https://stackoverflow.com/questions/4959975/generate-random-number-between-two-numbers-in-javascript

Leitakse kõik töötajad, kelle number on suurem võrdne leitud arvust. Tulemus sorteeritakse töötaja numbri alusel ja leitakse esimene rida. Eeliseks on see, et sorteeritavate ridade hulk on väiksem. Sorteerimise aluseks oleval veerul on indeks ja andmebaasisüsteem saab seda kasutada sorteerimise kiiremaks täitmiseks. See lahendus sobib ka olukorra jaoks kus töötajate numbrid ei tule järjest, vaid nende seas võib olla "auke".

```
SELECT TOP 1 *
FROM Tootaja
WHERE tootaja_kood>=Int(Rnd(tootaja_kood)*((SELECT Max(tootaja_kood)
FROM Tootaja)-(SELECT Min(tootaja_kood) FROM Tootaja))) + (SELECT
Min(tootaja_kood) FROM Tootaja)
ORDER BY tootaja_kood;
```

Vaese mehe otsingumootor – LIKE predikaadi või regulaaravaldiste kasutamine tekstiotsingute tegemiseks. Selle asemel võiks kasutada spetsiaalselt tekstiotsingu võimalusi, mida andmebaasisüsteem pakub (nt PostgreSQLis täisteksti otsing) või kolmandate osapoolte lisaprogramme.

Spagetipäring – ühe keerulise, halvasti loetava ja mõistetava lause kirjutamine mitme lihtsa omavahel seotud lause asemel. Keerulises lauses on lihtsam teha vigu – näiteks unustada kirja panna ühendamise tingimus ja nii tekitada korrutis.

Paljudes andmebaasisüsteemides saab loetavuse parandamiseks kasutada alampäringute WITH klauslis defineerimist. WITH klauslis võib defineerida mitu alampäringut. Iga järgmine alampäring saab viidata eelnevalt samas lauses defineeritud alampäringutele.

MS Accessis saab teha ühe keerulise lause asemel mitu lihtsamat, salvestada need (luua vaated) ja panna kokku lõpptulemus nende salvestatud päringute (vaadete) põhjal päringut tehes.

Näide (PostgreSQL).

Selle asemel, et kirjutada

```
SELECT *
FROM (SELECT deptno, Count(*) AS cnt
FROM Emp
GROUP BY deptno) AS foo
WHERE cnt>=ALL(SELECT Count(*) AS cnt FROM Emp GROUP BY
deptno);
```

saab kirjutada:

```
WITH number_of_res AS (
SELECT deptno, Count(*) AS cnt
FROM Emp
GROUP BY deptno)
SELECT * FROM number_of_res
WHERE cnt=(SELECT Max(cnt) AS m FROM number_of_res);
```

Veerunimede mitte väljatoomine.

Halb.

```
SELECT *
FROM Tootaja;
```

Päringu tulemusest kasutab väljakutsuja vaid töötaja koodi ja perenime. Andmebaasisüsteem peab tegema üleliigset tööd mittevajatavate andmete ülesleidmiseks. Nende andmete edastamine koormab arvutivõrku.

```
INSERT INTO Osakond
VALUES (100, 'Reklaamiosakond');
```

Kui tabelisse *Osakond* lisandub veerge või muutub veergude järjekord, siis lause enam ei tööta või töötab valesti.

Parem.

```
SELECT tootaja_kood, perenimi
FROM Tootaja;
```

```
INSERT INTO Osakond (osakonna_nr, osakonna_nimi)
VALUES (100, 'Reklaamiosakond');
```

Ousmane ja Xie (2019) kirjutavad, et selliste ja teiste samalaadsete vigade otsimiseks päringute tekstist saab kasutada masinõpet. Tegemist on teksti klassifitseerimise probleemiga ja närvivõrke saab treenida päringu liigitamiseks nulli või rohkemasse vigade klassi. Eksperimendis käsitleti edukalt täidetud SQL lausete hulka (1 miljon), millest üle 300000 olid SELECT laused. Eksperimendis käsitletigi ainult SELECT lauseid. Nad otsisid 16 tüüpilise antimustri esinemisi. Eksperimendi tulemuseks oli, et närvivõrkude abil vigade otsimisel oli täpsus 83.2 (100-st?) samas kui staatilist koodi analüüsi tegev programm SqlCheck, mis otsib samu probleeme, andis samade päringute puhul täpsuseks 80 (100-st?).

10. Relatsioonialgebra ja SQL

Järgnevas tabelis on kokkuvõtlikult nimetatud relatsioonialgebra operatsioone ja samas näidatud, kuidas taolisi operatsioone teostada SQLi abil.

Relatsioonialgebra operatsioon	***SQL lause näide***
Atribuudi ümbernimetamine	SELECT kood, nimi **AS uus_nimi** FROM Linn1;
Piirang	SELECT * FROM Tootaja **WHERE osakonna_nr=2 OR palk>1500;**
Projektsioon	SELECT **tootaja_kood, perenimi** FROM Tootaja;
Hulgateoreetiline summa	SELECT kood, nimi FROM Linn1 **UNION** SELECT kood, nimi FROM Linn2;
Hulgateoreetiline vahe	SELECT kood, nimi FROM Linn1 **EXCEPT** SELECT kood, nimi FROM Linn2;
Lõige	SELECT kood, nimi FROM Linn1 **INTERSECT** SELECT kood, nimi FROM Linn2;
Otsekorrutis	SELECT Linn1.kood, Linn1.nimi, Linn2.kood AS kood2, Linn2.nimi AS nimi2 **FROM Linn1, Linn2;**
Ühendamine (equijoin) (lisaks veel projektsioon)	SELECT Tootaja.*, Osakond.osakonna_nimi FROM Tootaja **INNER JOIN** Osakond ON Tootaja.osakonna_nr=Osakond.osakonna_nr; Ühendamine kasutades otsekorrutist ja piirangut: SELECT Tootaja.*, Osakond.osakonna_nimi FROM Tootaja, Osakond WHERE Tootaja.osakonna_nr=Osakond.osakonna_nr;
Välisühendamine (outer join) (lisaks veel projektsioon)	SELECT Tootaja.*, Osakond.osakonna_nimi FROM Tootaja **LEFT JOIN** Osakond ON Tootaja.osakonna_nr=Osakond.osakonna_nr;
Naturaalühendamine (natural join) (lisaks veel projektsioon)	SELECT perenimi, osakonna_nimi FROM Tootaja **NATURAL JOIN** Osakond;

Relatsioonialgebra operatsioon	***SQL lause näide***
Jagamine	Jagamise teostamiseks spetsiaalseid lausekonstruktsioone ei ole. Vajaliku tulemuse saab saavutada olemasolevaid lausekonstruktsioone kombineerides: SELECT R1.A, R1.B FROM R1 GROUP BY R1.A, R1.B HAVING Count(R1.C)=(SELECT Count(*) FROM R2);
Poolühendamine (semijoin)	SELECT DISTINCT Osakond.osakonna_nr, Osakond.osakonna_nimi FROM Osakond INNER JOIN Tootaja ON Osakond.osakonna_nr = Tootaja.osakonna_nr WHERE Tootaja.palk>1700;
Poolvahe leidmine (semidifference)	SELECT osakonna_nr, osakonna_nimi FROM Osakond WHERE osakonna_nr NOT IN (SELECT osakonna_nr FROM Tootaja WHERE osakonna_nr IS NOT NULL);
Laiendamine (lisaks veel projektsioon ja ümbernimetamine)	SELECT tootaja_kood, perenimi, palk, **Round(palk*15.6466,1)** AS palk_kroonides FROM Tootaja;
Summeerimine	SELECT osakonna_nr, **Avg**(palk) AS keskmine FROM Tootaja **GROUP BY** osakonna_nr;

11. Mõisted

Eesti keeles	Inglise keeles
Alampäring	Subquery, Nested query, Subselect
Descartesi ristkorrutis, Hulkade ristkorrutis, Otsekorrutis (relatsioonialgebra operatsioon)	Cartesian product, Unrestricted join
EXISTS predikaadis kasutatav alampäring, Eksistentsi kontrolliks kasutatav alampäring	Existential subquery
Funktsioon	Function
Hulgateoreetiline summa, Ühend (relatsioonialgebra operatsioon)	Union
Hulgateoreetiline vahe (relatsioonialgebra operatsioon)	Set difference
IN predikaadis kasutatav alampäring, Kuuluvuse kontrolliks kasutatav alampäring	Membership subquery
Klausel	Clause
Korreleeruv alampäring	Correlated subquery
Kokkuvõtmine, Rühmitamine (relatsioonialgebra operatsioon)	Summarize
Kokkuvõttefunktsioon, Grupifunktsioon, Agregaatfunktsioon	Aggregate function, Group function
Liitvõti	Composite key, Compound key
Lõige, Ühisosa (relatsioonialgebra operatsioon)	Intersection
Mittekorreleeruv alampäring	Non-correlated subquery
Naturaalühendamine (relatsioonialgebra operatsioon)	Natural join
Operaator	Operator
Predikaat	Predicate
Päring, Andmete otsimise lause	Query, Select statement

Eesti keeles	Inglise keeles
Rea alampäring	Row subquery
Relatsioonialgebra, Relatsioonalgebra	Relational algebra
Rohkem kui ühte alampäringut sisaldav päring	Compound nested query
Skalaarne alampäring	Scalar subquery
Sorteerimine	Sorting, Ordering
Sorteerimine kahanevalt	Sorting in descending order
Sorteerimine kasvavalt	Sorting in ascending order
Tabeli alampäring	Table subquery
Tabeli ühendamine iseendaga	Joining table to itself, Self join
Tavaühendamine	Inner join
Täielik välisühendamine	Full outer join
Vasakpoolne välisühendamine (relatsioonialgebra operatsioon)	(Left) outer join
Välisühendamine	Outer join
Ühendamine (relatsioonialgebra operatsioon)	Join
Ühe väärtuse tagastav alampäring, Singli alampäring	Singleton subquery
Ühendamine, kus võrdlusoperaatoriks ei ole võrdsuse kontrolli operaator	Non-equijoin
Ühendamine, kus võrdlusoperaatoriks on võrdsuse kontrolli operaator	Equijoin, Equi-join
Ühendamise ja ühendi kombinatsioon	Union join

12. Kasutatud materjalid

1. Andmebaasisüsteemi MS Access abifailid.
2. Cannan, S.J., Otten, G. A.M., 1992. *SQL - The Standard Handbook based on the new SQL standard (ISO 9075:1992(E))*.
3. Celcko, J., 2000. *SQL for smarties: advanced SQL programming. 2nd ed. Academic Press*. 553 p.
4. Connolly, T. M., Begg, C. E., 2001. *Database systems. A Practical Approach to Design, Implementation and Management*. Third Edition. Pearson Education. 1236 p.
5. Correlated Subqueries.[WWW] http://www.thunderstone.com/site/texisman/node98.html (14.03.2006)
6. Date, C. J., 2003. *An Introduction to Database Systems.* Eighth Edition. Addison Wesley. 983 p.
7. Date, C. J., 2009. *SQL and Relational Theory. How to Write Accurate SQL Code*, O'Reilly. 404 p.
8. Gagne, T., 2011. The Matryoshka Principle and Software Design. Toolbox.com, 17.11.2011. [WWW] http://it.toolbox.com/blogs/anything-worth-doing/the-matryoshka-principle-and-software-design-48938 (26.03.2016)
9. Gulutzan, P., Pelzer, T., 1999. *SQL-99 Complete, Really*. Miller Freeman. 1078 p.
10. Halpin, T., 2001. *Information Modeling and Relational Databases. From Conceptual Analysis to Logical Design*. San Francisco : Morgan Kaufman Publishers.
11. Karwin, B., 2010. *SQL Antipatterns. Avoiding the Pitfalls of Database Programming*. The Pragmatic Bookshelf.
12. Liiv, I.; Kuusik, R.; Võhandu, L., 2007. Conformity analysis with structured query language. 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases; Corfu Island, Greece; February 16-19, 2007. (Toim.) Long, C. A.; Mladenov, V. M.; Bojkovic, Z.. WSEAS, 2007, 187-189. [WWW] http://www.wseas.us/e-library/conferences/2007corfu/papers/540-155.pdf (21.03.2010)
13. Melton, J., ISO/IEC 9075-2:2003 (E) Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation). August, 2003.
14. Molinaro, A., 2005. *SQL Cookbook.* O'Reilly. 640 p.
15. Ousmane, A.R. and Xie, H., 2019. Detecting anti-patterns in SQL Queries using Text Classification Techniques. *International Journal of Advanced Engineering Research and Science*, Vol 6, No 4.Ousmane, A.R. and Xie, H., 2019. Detecting anti-patterns in SQL Queries using Text Classification Techniques. *International Journal of Advanced Engineering Research and Science*, Vol 6, No 4.
16. Tiit, E. M. Andmeanalüüs I; 3. Loeng. Pidev tunnus. Variatsioonrida ja kvantiilid. 2001. [WWW] http://www.ms.ut.ee/oppetoo/AA1/AA1_Loeng3.htm (28.02.2003)

17. Tropashko, V., 2006. *SQL Design Patterns. The Expert Guide to SQL Programming*. Rampant Techpress. 254 p.