Kohustusliku iseseisva harjutamise ülesande 2 kommenteeritud vastused

Sisukord

1. Sissejuhatus

Teema: SELECT laused mitme tabeli põhjal.

Kasutage MS Accessi andmebaasi *Magajate_Systeem.mdb*. Salvestage kõik tehtud SQL laused. Hiljem pannakse ülesannete lahendused ka veebikeskkonda välja.

Kui ülesandes on öeldud nt “leia magajad...” või “leia asemed...”, siis juhul, kui pole öeldud teisiti, tuleb leida selle olemi kõik andmed (kõik veerud samanimelisest tabelist).

Ülesannetes esitatud tulemuse näited ei pruugi langeda kokku vastustega, mida saadakse hetkel andmebaasis olevate andmete põhjal.

Jaotistes 2–5 ning jaotise 6 alguses on võrreldud SQL lauseid erinevates andmebaasisüsteemides. Ülesannete lahenduste juures kasutatakse järgmist tähistust.

- **A** – töötab andmebaasisüsteemis *MS Access 2013*
- **P** – töötab andmebaasisüsteemis *PostgreSQL 9.3*
- **O** – töötab andmebaasisüsteemis *Oracle Enterprise Edition 12c Release 1*
- **S** – *töötab andmebaasisüsteemis Ocelot (versioon, mis avaldati 14. september 2001) ja seega peaks vastama SQL standardile.*

2. Funktsioonide kasutamine

1. **Leia hetke kuupäev ja kellaaeg üheskoos (arvuti kella poolt näidatav aeg). Leia ainult hetke kuupäev (ilma kellaajata). Leia ainult hetke kellaaeg. Päringu tulemuses peavad veergude nimed olema vastavalt *hetke_kuupaev_ja_kellaaeg*, *hetke_kuupäev* ja *hetke_kellaaeg*.**
   Vihje: Kasutage funktsioone *Now()*, *Date()*, *Time()*.

```
SELECT Now() AS hetke_kuupaev_ja_kellaaeg, Date() AS hetke_kuupäev,
    Time() AS hetke_kellaaeg;                                    (A)
```

```
SELECT CURRENT_DATE AS hetke_kuupäev, CURRENT_TIME AS
    hetke_kellaaeg, CURRENT_TIMESTAMP AS
    hetke_kuupaev_ja_kellaaeg;                                   (P)
```

```
SELECT CURRENT_DATE AS hetke_kuupäev, CURRENT_DATE AS
    hetke_kellaaeg, CURRENT_TIMESTAMP AS hetke_kuupaev_ja_kellaaeg
FROM Dual;                                                       (S)
```

```
SELECT CURRENT_DATE AS hetke_kuupäev, to_char(CURRENT_DATE,
    'hh:mm:ss') AS hetke_kellaaeg, CURRENT_TIMESTAMP AS
    hetke_kuupaev_ja_kellaaeg
FROM Dual;                                                       (O)
```

PostgreSQLis on funktsioon now() vaikimisi olemas. Nii Oracles, PostgreSQLis kui ka Ocelotis saab kirjutada funktsioone ja seega on võimalik kasutajal soovi korral ise taolised funktsioonid luua.

3. Tabelite ühendamine

2. **Leia tabelis *Magaja* ja *Magamine* olevate andmete otsekorrutis.**

Tulemuses on kõik read ühest tabelist on ühendatud kõigi teiste ridadega teisest tabelist. Kui tabelis Magaja on 18 ja tabelis Magamine on 52 rida siis päringu tulemuseks saadavas virtuaalses tabelis on 18*52=936 rida.

Korrutisel on mõte näiteks siis, kui soovida päringu tulemusena leida kõikvõimalikke olemite paare.

```
SELECT Magaja.*, Magamine.*
FROM Magaja, Magamine;                                       (APOS)
```

```
SELECT *
FROM Magaja, Magamine;                                       (APOS)
```

```
SELECT *
FROM Magaja CROSS JOIN Magamine;                             (POS)
```

SQL standardi järgi ei tohi olla baastabelis mitu sama nimega veergu. Kuid kui Te teete päringu, siis selles on SQLis lubatud mitu sama nimega veergu. See on vastuolus relatsioonilise mudeli reeglitega, sest relatsioonis ei tohi olla mitu sama nimega atribuuti.

Kui päringu tulemusena tekkivas virtuaalses tabelis on mitu ühesuguse nimega veergu, siis pannakse MS Accessis veeru nime ette tabeli nimi (*Tabeli_nimi.veeru_nimi*), kust see veerg on võetud (juhul kui veeru nimi ei ole

alias). Teistes vaadeldavates andmebaasisüsteemides esitatakse päringu tulemusena tabel, kus on mitu ühesuguse nimega veergu.

Näide PostgreSQLis. Tehes päringu

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

on tulemuseks järgnev tabel.

```
 perenimi | perenimi
----------+----------
 Vaarikas | Vaarikas
 Jaan     | Jaan
 Ta?mik   | Ta?mik
 Juurikas | Juurikas
 Karu     | Karu
 Vaarikas | Vaarikas
 Jaaniste | Jaaniste
 Mändl    | Mändl
 Jansen   | Jansen
 Saan]    | Saan]
 Mets     | Mets
 Pikk     | Pikk
 V*%_l    | V*%_l
 Meri     | Meri
 Lepp     | Lepp
 Roi      | Roi
 Jaan     | Jaan
 Raag     | Raag
(18 rows)
```

Kuid üritades selle tabeli põhjal omakorda päringut teha, on tulemuseks veateade.

```
SELECT perenimi
FROM (SELECT perenimi, perenimi FROM Magaja) AS foo;
```

Veateade.
ERROR: column reference "perenimi" is ambiguous
LINE 1: SELECT perenimi

Siit moraal – päringud tuleks kirjutada nii, et päringu tulemuses ei oleks mitu sama nimega veergu.

3. **Leia magamiste andmed. Päringu tulemuses peaksid olema järgmised veerud: magaja identifikaator, magaja ees- ja perenimi ühe stringina, magamise alguse aeg, magamise kestus, aseme identifikaator ja aseme nimi. Tee päring kahel viisil, kirjutades ühendamise tingimuse esimesel juhul WHERE ja teisel juhul FROM klauslisse. (NB! SQL kontrolltöös peab tabelite ühendamist oskama teha mõlemal viisil).**

Ühendamine tähendab, et meid ei huvita otsekorrutise tulemusest mitte kõik kombinatsioonid (mitte kõik magajate ja magamiste paarid) vaid ainult teatud tingimusele vastavad. Antud juhul on tabelite *Magaja* ja *Magamine* vahele loodud seos veergude *magaja_id* abil ja otsekorrutise tulemusest on vajalikud vaid sellised read, kus nendes veergudes on väärtused võrdsed.

Magaja. magaja_id	Magamine. magaja_id	eesnimi	perenimi	algus	ase_id
1	2	Jaan	Vaarikas	16.04.2002 21:00:00	1
2	2	Andres	Jaan	16.04.2002 21:00:00	1
3	2	Taimi	Ta?mik	16.04.2002 21:00:00	1
4	2	Juhan	Juurikas	16.04.2002 21:00:00	1
5	2	Kati	Karu	16.04.2002 21:00:00	1
6	2	Juhan	Vaarikas	16.04.2002 21:00:00	1
7	2	Jaan	Jaaniste	16.04.2002 21:00:00	1
8	2	Jüri	Mänd1	16.04.2002 21:00:00	1
9	2	Anu	Jansen	16.04.2002 21:00:00	1
10	2	Jaan	Saan]	16.04.2002 21:00:00	1
11	2		Mets	16.04.2002 21:00:00	1
12	2	Taavi	Pikk	16.04.2002 21:00:00	1
13	2	Avo	V*_l	16.04.2002 21:00:00	1
14	2	Andres	Meri	16.04.2002 21:00:00	1
15	2	Valli	Lepp	16.04.2002 21:00:00	1
16	2	Roby	Roi	16.04.2002 21:00:00	1
17	2	Rudolf	Jaan	16.04.2002 21:00:00	1
19	2	Reet	Raag	16.04.2002 21:00:00	1
1	5	Jaan	Vaarikas	16.06.2002 21:00:00	1
2	5	Andres	Jaan	16.06.2002 21:00:00	1
3	5	Taimi	Ta?mik	16.06.2002 21:00:00	1
4	5	Juhan	Juurikas	16.06.2002 21:00:00	1
5	5	Kati	Karu	16.06.2002 21:00:00	1
6	5	Juhan	Vaarikas	16.06.2002 21:00:00	1
7	5	Jaan	Jaaniste	16.06.2002 21:00:00	1
8	5	Jüri	Mänd1	16.06.2002 21:00:00	1
9	5	Anu	Jansen	16.06.2002 21:00:00	1

Magaja.magaja_id	Magamine.magaja_id	eesnimi	perenimi	algus	ase_id
10	5	Jaan	Saan]	16.06.2002 21:00:00	1
11	5		Mets	16.06.2002 21:00:00	1
12	5	Taavi	Pikk	16.06.2002 21:00:00	1
13	5	Avo	V*_l	16.06.2002 21:00:00	1
14	5	Andres	Meri	16.06.2002 21:00:00	1
15	5	Valli	Lepp	16.06.2002 21:00:00	1
16	5	Roby	Roi	16.06.2002 21:00:00	1
17	5	Rudolf	Jaan	16.06.2002 21:00:00	1
19	5	Reet	Raag	16.06.2002 21:00:00	1
...	...	...	...	...	...

a) Tabelite ühendamise tingimuste kirjutamine WHERE klauslisse. See on vanem süntaks.

```
SELECT Magaja.magaja_id, Magaja.eesnimi & ' ' & Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM Magamine, Magaja, Ase
WHERE (Magamine.magaja_id=Magaja.magaja_id) AND
    (Magamine.ase_id=Ase.ase_id);
                                                              (A)
```

Võib ka kirjutada sulgudeta:

```
SELECT Magaja.magaja_id, Magaja.eesnimi & ' ' & Magaja.perenimi AS
    nimi, Ase.ase_id, Ase.nimi
FROM Magamine, Magaja, Ase
WHERE Magamine.magaja_id=Magaja.magaja_id AND
    Magamine.ase_id=Ase.ase_id;

                                                              (A)
```

b) Uuema süntaksi kohaselt (võimalik alates SQL standardi versioonist SQL-1992) võib ühendamise tingimused kirjutada ka FROM klauslisse.

```
SELECT Magaja.magaja_id, Magaja.eesnimi & ' ' & Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM Magaja INNER JOIN (Ase INNER JOIN Magamine ON Ase.ase_id
    = Magamine.ase_id) ON Magaja.magaja_id = Magamine.magaja_id;
                                                              (A)
```

Ühendamise tingimuse kirjutamine FROM klauslisse parandab loetavust kui WHERE klauslisse soovitakse lisada täiendavaid tingimusi. MS Accessis (2010) võib WHERE klausel sisaldada kuni 99 AND operaatori poole

pöördumist (http://office.microsoft.com/en-us/access-help/access-2010-specifications-HA010341462.aspx).

Ühendamise tingimuste järjekord lauses ei muuda päringu lõpptulemust, kuid *võib* mõjutada andmebaasisüsteemi poolt lause täitmiseks koostatava täitmisplaani koostamist.

SQL lubab päringu tulemusena moodustavas tabelis (nimeta tabelis) mitut samanimelist veergu, kuid selliste päringute kirjutamine on halb stiil. See ei anna midagi juurde, vaid tekitab ainult segadust. **NB!** Tehes järgneva päringu, mille tulemuses on kaks samanimelist veergu (millest ühe nimi on määratud aliasega),

```
SELECT Magaja.magaja_id, Magaja.eesnimi & ' ' & Magaja.perenimi AS
    nimi, Ase.ase_id, Ase.nimi
FROM Magamine, Magaja, Ase
WHERE (Magamine.magaja_id=Magaja.magaja_id) AND
    (Magamine.ase_id=Ase.ase_id);                                    (A)
```

hakkab MS Access kummaliselt käituma. Magaja eesnimest ja perenimest kokku pandud nime (veerus *nimi*) kasutajale ei näidata. Tabeli *Ase* veeru *nimi* põhjal tekkiva veeru nimi on *Ase.nimi*.

Oletame, et FROM klauslis viidatakse kahele või rohkemale tabelile *t1,...tn*. Oletame, et vähemalt kahes tabelis tabelite hulgast *t1,..., tn* on veerg nimega *v*. Kui SQL lauses soovitakse viidata veerule *v*, siis peab täpsustama, millise tabeli veergu mõeldakse. Selleks tuleb veeru nimi esitada kujul *ti.v*, kus *ti* on tabeli nimi, milles *v* sisaldub (see on veeru *qualified name*). Kui veeru nimi on unikaalne üle kõikide tabelite *t1,..., tn*, siis piisab veerule viitamiseks nimest *v*.

NB! SQL lubab kasutada ka järgneva süntaksiga päringuid sama ülesande lahendamiseks (MS Accessis (2013) neid kasutada ei saa)

```
SELECT Magaja.magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM Magamine CROSS JOIN Magaja CROSS JOIN Ase
WHERE (Magamine.magaja_id=Magaja.magaja_id) AND
    (Magamine.ase_id=Ase.ase_id);
                                                                     (POS)
```

Ühendamine kasutades uut süntaksi (MS Accessis ei saa seda lauset käivitada tänu "||" operaatori kasutamisele ning CROSS JOIN operaatori kasutamisele).

```
SELECT  Magaja.magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM (Magamine INNER JOIN Magaja ON
    Magamine.magaja_id=Magaja.magaja_id) INNER JOIN Ase ON
    Magamine.ase_id=Ase.ase_id;
```

(PO)

Võib ka kirjutada ilma sulgudeta.

```
SELECT Magaja.magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM Magamine INNER JOIN Magaja ON
    Magamine.magaja_id=Magaja.magaja_id INNER JOIN Ase ON
    Magamine.ase_id=Ase.ase_id;
```

(POS)

Ühendamise tüübi "INNER" võib jätta kirjutamata. See on vaikimisi valik.

```
SELECT  Magaja.magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM (Magamine JOIN Magaja ON
    Magamine.magaja_id=Magaja.magaja_id) JOIN Ase ON
    Magamine.ase_id=Ase.ase_id;
```

(PO)

```
SELECT  Magaja.magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM Magamine JOIN Magaja ON
    Magamine.magaja_id=Magaja.magaja_id JOIN Ase ON
    Magamine.ase_id=Ase.ase_id;
```

(POS)

Veel üks näide uuest süntaksist.

```
SELECT  magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM (Magamine JOIN Magaja USING (magaja_id)) JOIN Ase USING
    (ase_id);
```

(PO)

Järgnev päring aga ei anna soovitud tulemust, sest nii tabelis *Magaja* kui ka *Magamine* on samanimeline veerg *kommentaar*, mis ei sisalda ühendamiseks vajalikke andmeid, kuid mida andmebaasisüsteem üritab ikkagi kasutada.

```
SELECT Magaja.magaja_id, Magaja.eesnimi || ' ' || Magaja.perenimi AS
    magaja_nimi, Ase.ase_id, Ase.nimi AS aseme_nimi, Magamine.algus,
    Magamine.kestus, Magamine.kommentaar
FROM Magaja NATURAL JOIN Magamine NATURAL JOIN Ase;
```

4. **Kirjuta üks eelmises ülesandes loodud päring ümber, kasutades päringus tabelitele viitamiseks aliasi (ingl *correlation names*) (aliased võib ise valida ning need võiksid olla lühemad, kui tabelite pärisnimed).**

```
SELECT M.magaja_id, eesnimi & ' ' & perenimi AS magaja_nimi, A.ase_id,
    nimi AS aseme_nimi, algus, kestus, Ma.kommentaar
FROM Magamine AS Ma, Magaja AS M, Ase AS A
WHERE (Ma.magaja_id=M.magaja_id) AND (Ma.ase_id=A.ase_id);      (A)
```

```
SELECT M.magaja_id, M.eesnimi & ' ' & M.perenimi AS magaja_nimi,
    A.ase_id, A.nimi AS aseme_nimi, Ma.algus, Ma.kestus, Ma.kommentaar
FROM Magaja AS M INNER JOIN (Ase AS A INNER JOIN Magamine AS
    MA ON A.ase_id = Ma.ase_id)  ON M.magaja_id = Ma.magaja_id;
    (A)
```

Aliasi kasutatakse antud juhul SQL lause lühendamiseks. Need kehtivad ainult antud SQL-lause piirides. Seega võib sama aliast kasutada mistahes tabeli jaoks mistahes teises SQL lauses. Selle lause abil ei nimetata andmebaasis tabeleid ümber (selleks on ALTER TABLE lause).

Taolisi aliasi läheb kindlasti vaja, kui kasutatakse ***korreleeruvat alampäringut*** ja nii põhipäringus kui ka alampäringus pöördutakse sama tabeli poole. Samuti läheb aliasi vaja, kui kasutatakse ***self-joini*** e. tabeli ühendamist iseendaga. Aliase defineerimisel võib, kuid ei pea, kasutama AS fraasi. Seega on legaalne ka järgnev lause.

```
SELECT M.magaja_id, eesnimi & ' ' & perenimi AS magaja_nimi, A.ase_id,
    nimi AS aseme_nimi, algus, kestus, Ma.kommentaar
FROM Magamine Ma, Magaja M, Ase A
WHERE (Ma.magaja_id=M.magaja_id) AND (Ma.ase_id=A.ase_id);      (A)
```

```
SELECT M.magaja_id, eesnimi || ' ' || perenimi AS magaja_nimi, A.ase_id,
    nimi AS aseme_nimi, algus, kestus, Ma.kommentaar
FROM Magamine Ma, Magaja M, Ase A
WHERE (Ma.magaja_id=M.magaja_id) AND (Ma.ase_id=A.ase_id);   (POS)
```

NB! Date ja Darwen (1997) märgivad, et SQL standardi kohaselt kirjeldatakse FROM klauslis jada- ehk vahemikmuutujad (ingl *range variable*), mille lubatud väärtusteks on tabelite read. Nende muutujate põhjal leitakse päringu tulemus. Igal muutujal peab olema nimi ning vajadusel määrab selle andmebaasisüsteem. Seega kui käivitada päring:

```
SELECT *
FROM Magaja;
```

, siis täidab andmebaasisüsteem tegelikult päringu:

```
SELECT *
FROM Magaja AS Magaja;                                    (APS)
```

```
SELECT *
FROM Magaja Magaja;                                       (O)
```

Magaja on muutuja nimi. Nagu näete, kasutatakse muutuja vaikimisi nimena tabeli enda nime. SQL standard kasutab sellisele nimele viitamiseks terminit *correlation name*. Seega täidetakse rangelt võttes antud päring muutuja *Magaja* põhjal, mille väärtuse moodustavad tabelis *Magaja* hetkel olevad andmed.

5. **Ühenda päringu tulemuses magajate ja nende mõõtmiste andmed. Päringu tulemuses peaksid olema järgmised veerud: magaja identifikaator, magaja ees- ja perenimi ühe stringina, mõõtmise aeg, pikkus, kaal. Kui magajat pole mõõdetud, peavad päringu tulemuses ikkagi olema tema isikuandmed (identifikaator ja nimi). Mõõtmiste andmete kohal on siis väljad tühjad.**

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja LEFT OUTER JOIN Mootmine ON Magaja.magaja_id =
    Mootmine.magaja_id;
                                                          (A)
```

Tulemuses on kollasega tähistatud read, mis on tulemuses tänu "tavalise" ühendamise asemel välisühendamise (ingl *outer join*) kasutamisele. Leitakse ka sellised magajad, kelle kohta pole tehtud ühtegi mõõtmist. Süntaksi järgi Tabel1 LEFT OUTER JOIN Tabel 2 – Leitakse kõik read tabelist *Tabel1*, sest tema nimi on ***vasakul*** pool (ingl ***left***) fraasi "LEFT JOIN".

magaja_id	nimi	mootmise_aeg	pikkus	kaal
1	Jaan Vaarikas	14.01.2004 15:00:00	190	85
1	Jaan Vaarikas	14.02.2004 18:30:00	190	88
1	Jaan Vaarikas	27.02.2004 15:41:19	190	87
2	Andres Jaan	14.01.2004 15:20:00	185	84
2	Andres Jaan	14.02.2004 18:45:00	186	84
3	Taimi Ta?mik	14.01.2004 15:40:00	165	54
3	Taimi Ta?mik	14.02.2004 18:00:00	165	54
4	Juhan Juurikas	14.01.2004 16:00:00	179	70
5	Kati Karu	14.01.2004 16:10:00	173	53
6	Juhan Vaarikas	14.01.2004 16:20:00	205	100
7	Jaan Jaaniste	14.01.2004 16:30:00	200	89
8	Jüri Mänd1	14.01.2004 16:40:00	210	98

magaja_id	nimi	mootmise_aeg	pikkus	kaal
9	Anu Jansen	14.01.2004 16:50:00	165	69
10	Jaan Saan]	14.01.2004 17:00:00	182	135
11	Mets	14.01.2004 17:10:00	167	70
12	Taavi Pikk	14.01.2004 17:20:00	199	180
12	Taavi Pikk	27.02.2004 15:49:21	200	180
13	Avo V*_I			
14	Andres Meri			
15	Valli Lepp			
16	Roby Roi			
17	Rudolf Jaan			
19	Reet Raag			

NB! Sõna OUTER ei ole kohustuslik. Seetõttu on samaväärne ka päring:

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja LEFT JOIN Mootmine ON Magaja.magaja_id =
    Mootmine.magaja_id;
                                                    (A)
```

Samasuguse tulemuse annab ka päring:

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Mootmine RIGHT JOIN Magaja ON Magaja.magaja_id =
    Mootmine.magaja_id;
                                                    (A)
```

Antud juhul võetakse kõik read tabelist, mille nimi jääb *paremale* poole fraasist "RIGHT JOIN". Kuna tabelite nimede järjekord on muutunud, on tulemus sama.

Võrrelge tulemust.

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja INNER JOIN Mootmine ON Magaja.magaja_id =
    Mootmine.magaja_id;
```

magaja_id	nimi	mootmise_aeg	pikkus	kaal
1	Jaan Vaarikas	14.01.2004 15:00:00	190	85
1	Jaan Vaarikas	14.02.2004 18:30:00	190	88
1	Jaan Vaarikas	27.02.2004 15:41:19	190	87

magaja_id	nimi	mootmise_aeg	pikkus	kaal
2	Andres Jaan	14.01.2004 15:20:00	185	84
2	Andres Jaan	14.02.2004 18:45:00	186	84
3	Taimi Ta?mik	14.01.2004 15:40:00	165	54
3	Taimi Ta?mik	14.02.2004 18:00:00	165	54
4	Juhan Juurikas	14.01.2004 16:00:00	179	70
5	Kati Karu	14.01.2004 16:10:00	173	53
6	Juhan Vaarikas	14.01.2004 16:20:00	205	100
7	Jaan Jaaniste	14.01.2004 16:30:00	200	89
8	Jüri Mänd1	14.01.2004 16:40:00	210	98
9	Anu Jansen	14.01.2004 16:50:00	165	69
10	Jaan Saan]	14.01.2004 17:00:00	182	135
11	Mets	14.01.2004 17:10:00	167	70
12	Taavi Pikk	14.01.2004 17:20:00	199	180
12	Taavi Pikk	27.02.2004 15:49:21	200	180

```
SELECT Magaja.magaja_id, eesnimi || ' ' || perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja LEFT JOIN Mootmine ON Magaja.magaja_id =
    Mootmine.magaja_id;
```

(POS)

Gulutzan ja Pelzer (1999) nimetavad välisühendamise operatsiooni puuduseid.

- Päringu tulemusest ei saa aru, kas seal on NULLid selle pärast, et päringu aluseks oleva tabeli rea mõnes väljas pole andmeid või hoopis selle pärast, et viiakse läbi välisühendamise operatsioon.
- Mõned andmebaasisüsteemid kasutavad sellise operatsiooni puhul spetsiifilist, standardiseerimata süntaksi.
- Andmete leidmine on aeglasem kui INNER JOIN operaatori kasutamise puhul. Järelikult välisühendamist ei tohiks kasutada "igaks juhuks" vaid ainult siis, kui "tavaline" ühendamine (inner join) ei anna soovitud tulemust.

NB! Oletame, et meil on järgnev ülesanne.

Ühenda päringu tulemuses magajate ja nende **2004. aastal toimunud mõõtmiste andmed**. Päringu tulemuses peaksid olema järgmised veerud: magaja identifikaator, magaja ees- ja perenimi ühe stringina, mõõtmise aeg, pikkus, kaal. Kui magajat pole aastal 2004 kordagi mõõdetud, peavad päringu tulemuses ikkagi olema tema isikuandmed (identifikaator ja nimi). Mõõtmiste andmete kohal on siis väljad tühjad.

Vale lahendus.

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja LEFT JOIN Mootmine ON Magaja.magaja_id =
    Mootmine.magaja_id
WHERE Year(Mootmine.mootmise_aeg)=2004;                    (A)
```

Selle päringu tulemuses on andmed vaid selliste magajate kohta, kelle kohta on teada vähemalt ühe aastal 2004 toimunud mõõtmise andmed. See lahendus andis vale tulemuse, sest kõigepealt viidi läbi ühendamise operatsioon. Ühendamise tulemusele rakendati piirangu operatsioon ning elimineeriti kõik read, kus tingimus *Year(Mootmine.mootmise_aeg)=2004* ei ole täidetud (kaasa arvatud read, kus mõõtmise aeg puudus).

Vale lahendus.

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja LEFT JOIN Mootmine ON Magaja.magaja_id =
    Mootmine.magaja_id
WHERE Year(Mootmine.mootmise_aeg)=2004 OR
    Mootmine.mootmise_aeg IS NULL;                         (A)
```

Selle päringu tulemuses ei ole andmeid magajate kohta, kellel on üks või rohkem seotud mõõtmist, kuid ükski mõõtmine ei ole toimunud 2004. aastal.

Kuidas saada päringu tulemusse andmed magajate kohta, kelle puhul ei ole registreeritud ühtegi aastal 2004 toimunud mõõtmist?

Õige lahendus.

```
SELECT Magaja.magaja_id, eesnimi & ' ' & perenimi AS nimi,
    mootmise_aeg, pikkus, kaal
FROM Magaja LEFT JOIN (SELECT * FROM Mootmine WHERE
    Year(Mootmine.mootmise_aeg)=2004) AS Mootmine ON
    Magaja.magaja_id = Mootmine.magaja_id;                 (A)
```

Selle lahenduse korral leiti alampäringuga 2004. aastal toimunud mõõtmised ning selle operatsiooni tulemusena leitud andmed ühendati tabelis *Magaja* olevate andmetega. Välisühendamise kasutamine tagab, et kui magajale pole 2004. aastal ühtegi mõõtmist tehtud, siis võetakse ikkagi tema isikuandmed päringu tulemusse sisse.

6. **Tehke päring, millega leiate aastal 2004 tehtud mõõtmised, mille käigus mõõdeti magajat, kelle eesnimi on Jaan ja perenimi on Vaarikas. Päringu tulemuses peavad olema andmed kõigist tabeli *Mootmine* veergudest. *Vihje*: Kasutage funktsiooni *Year*, et leida aastal 2004 toimunud mõõtmised.**

```
SELECT Mo.*
FROM Magaja AS Ma INNER JOIN Mootmine AS Mo ON
Ma.magaja_id=Mo.magaja_id
WHERE Ma.eesnimi='Jaan' AND Ma.perenimi='Vaarikas' AND
Year(Mo.mootmise_aeg)=2004;
```
(A)

```
SELECT Mo.*
FROM Magaja Ma INNER JOIN Mootmine Mo ON
Ma.magaja_id=Mo.magaja_id
WHERE Ma.eesnimi='Jaan' AND Ma.perenimi='Vaarikas' AND Extract
(YEAR FROM mootmise_aeg)=2004;
```
(PO)

Ülesande lahendamiseks võib ka kasutada alampäringuid. Alampäringute abil leitakse Jaan Vaarika *magaja_id* ning aastal 2004 toimunud mõõtmised. Seejärel need andmed ühendatakse.

```
SELECT Mo.*
FROM (SELECT magaja_id FROM Magaja WHERE eesnimi='Jaan' AND
perenimi='Vaarikas') AS Ma INNER JOIN (SELECT * FROM Mootmine
WHERE Year(mootmise_aeg)=2004) AS Mo ON
Ma.magaja_id=Mo.magaja_id;
```
(A)

```
SELECT Mo.*
FROM (SELECT magaja_id FROM Magaja WHERE eesnimi='Jaan' AND
perenimi='Vaarikas') Ma INNER JOIN (SELECT * FROM Mootmine WHERE
Extract (YEAR FROM mootmise_aeg)=2004) Mo ON
Ma.magaja_id=Mo.magaja_id;
```
(PO)

4. Grupeerimine

7. Leia iga magaja kohta tema kõige vähem ja kõige kauem kestnud magamise kestus ning keskmine magamise kestus. Esita tulemuses ka magaja identifikaator, eesnimi ja perenimi.

Selle ülesande lahendamiseks tuleb kaks tabelit omavahel ühendada ja kasutada grupeerimist.

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Max(kestus) AS
max_pikkus, Min(kestus) AS min_pikkus, Avg(kestus) AS
keskm_pikkus
FROM Magaja INNER JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi;                (APO)
```

NB! Pange tähele, et päringus kasutatakse grupeerimisel ka magaja unikaalset identifikaatorit. See on vajalik, sest võib olla mitu ühesuguse nimega magajat. Ilma identifikaatorit grupeerimisel kasutamata võetaks need kokku üheks grupiks (ehki seal taga on erinevad inimesed). Kõik veerud, mis on nimetatud SELECT klauslis ja millele pole rakendatud kokkuvõttefunktsiooni, tuleb üles lugeda ka GROUP BY klauslis. Nendes veergudes olevate väärtuste põhjal luuaksegi ridade grupid.

Andmebaasis, mis oli loodud andmebaasisüsteemis Ocelot oli veerg *kestus* intervalli tüüpi. See päring ei läinud käima, sest seal ei tohtinud Avg funktsiooni rakendada intervalli tüüpi andmeid sisaldavale veerule. Gulutzan ja Pelzer (1999) kohaselt on see SQL standardis siiski lubatud. Tulemuses peaks intervall olema sama täpsusega kui argumendiks olnud veerus.

PostgreSQL ja Oracle andmebaasisüsteemides on võimalik seda ülesannet ka lahendada kasutades *analüütilisi funktsioone.* Selliste funktsioonide kasutamise võimalus on ette nähtud ka hetkel kehtivas SQL standardi versioonis (SQL:2011). Järgnevas artiklis kirjeldatakse analüütilisi funktsioone Oracle näitel: http://www.orafaq.com/node/55

Nagu näete, puudub nüüd päringus GROUP BY klausel.

```
SELECT DISTINCT Magaja.magaja_id, eesnimi, perenimi,
Max(kestus) OVER (PARTITION BY Magamine.magaja_id) AS max_pikkus,
Min(kestus) OVER (PARTITION BY Magamine.magaja_id) AS min_pikkus,
Avg(kestus) OVER (PARTITION BY Magamine.magaja_id) AS
keskm_pikkus
FROM Magaja INNER JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id;                                          (PO)
```

8. **Leia selliste magajate identifikaator, eesnimi, perenimi ja magamiste arv, kellega seotud magamiste arv on suurem kui 4. Sorteeri tulemus magamiste arvu järgi kasvavalt.**

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Count(*) AS magamiste_arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi
HAVING Count(*)>4
ORDER BY Count(*);                                                    (APO)
```

HAVING klauslisse kirjutatud tingimus (predikaat) piirab nende GROUP BY abil moodustatud gruppide andmete väljastamist, mis ei vasta päringu tingimustele. See võimaldab piirata väljastatavaid ridu vastavalt kokkuvõttefunktsiooni tulemustele.

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Count(*) AS magamiste_arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi
HAVING Count(*)>4
ORDER BY magamiste_arv;                                               (POS)
```

NB! Tüüpiline viga. HAVING ja ORDER BY klauslites ei saa MS Accessis kasutada aliasi. HAVING klauslis aliase kasutamist ei näe ette SQL standard. ORDER BY klauslis lubab SQL standard aliasi kasutada, aga MS Accessis pole see lubatud.

Ülesande lahendaks ka päring, kus FROM klauslis oleva alampäringuga leitakse kõigi maganud magajate magamiste arv ja peapäringus rakendatakse sellele kitsendus.

```
SELECT *
FROM (SELECT Magaja.magaja_id, eesnimi, perenimi, Count(*) AS
magamiste_arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi) AS mag_arv
WHERE magamiste_arv>4
ORDER BY magamiste_arv;                                               (APS)
```

Oracles ei tohi FROM klauslis alampäringu abil kirjeldatud tabelile nime andmisel kasutada sõna "AS". Saate veateate "ERROR at line 4:
ORA-00933: SQL command not properly ended".

Teistes vaadeldavates andmebaasisüsteemides ei ole see oluline.

```
SELECT *
FROM (SELECT Magaja.magaja_id, eesnimi, perenimi, Count(*) AS
magamiste_arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi)  mag_arv
WHERE magamiste_arv>4
ORDER BY magamiste_arv;                                    (APOS)
```

Tänu sellele, et tänapäeval saab SQLis kasutada FROM klauslis alampäringut, on HAVING klausel muutunud sisuliselt üleliigseks. Ülesanne on võimalik lahendada ka nii, et lauses GROUP BY klauslit ei kasutata.

Samas, GROUP BY ja HAVING klausleid ei saa SQList eemaldada, sest paljud süsteemid juba kasutavad selliseid klausleid sisaldavaid lauseid.

Järgnevalt esitan lahenduse, mis kasutab SELECT klauslis asuvat korreleeruvat alampäringut. ülesande lahenduses ei kasutata GROUP BY klauslit.

```
SELECT *
FROM (SELECT Magaja.magaja_id, eesnimi, perenimi, (SELECT  Count(*)
AS arv FROM Magamine WHERE
Magaja.magaja_id=Magamine.magaja_id)  AS magamiste_arv
FROM Magaja) mag_arv
WHERE magamiste_arv>4
ORDER BY magamiste_arv;                                    (APOS)
```

Antud näites on üks alampäring teise sees. Sisemine on korreleeruv alampäring, mis leiab magajatega seotud magamiste arvu.

Alati pole koos kokkuvõttefunktsiooni kasutamisega vaja HAVING klauslit kasutada. Näiteks olgu ülesanne: **"Leia sellised magajad, kelle magamise maksimaalne kestus on suurem kui 30 minutit. Näita tulemuses magaja identifikaatorit, eesnime, perenime ja tema magamise maksimaalset kestust. Sorteeri tulemus maksimaalse kestuse järgi kasvavalt".**

```
SELECT eesnimi, perenimi, Max(kestus) AS pikim_magamine
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
WHERE kestus>30
GROUP BY Magaja.magaja_id, eesnimi, perenimi
ORDER BY Max(kestus);                                      (APO)
```

Kui kirjutada WHERE klauslisse tingimus **kestus>30**, siis sellega juba jäetakse vaatluse alt välja read, kus magamise kestus on 30 minutit või vähem.

Ülesande võib lahendada ka nii, et loobuda üldse WHERE klausli kasutamisest ja kirjutada täiendav kitsendus koos ühendamise tingimusega FROM klauslisse.

```
SELECT eesnimi, perenimi, Max(kestus) AS pikim_magamine
FROM Magaja INNER JOIN Magamine ON
(Magaja.magaja_id=Magamine.magaja_id AND  Magamine.kestus>30)
GROUP BY Magaja.magaja_id, eesnimi, perenimi
ORDER BY Max(kestus);                                          (APO)
```

See ei ole siiski soovitav lahendus, sest sarnaselt ühendamise vanema süntaksiga muutub nüüd FROM klausel (WHERE klausli asemel) liiga ülekoormatuks. MS Access ei luba sellist päringut disainivaates vaadata.

Andmebaasis, mis oli loodud andmebaasisüsteemis Ocelot, oli veerg *kestus* intervalli tüüpi. Seega tuleb kirjutada päring:

```
SELECT eesnimi, perenimi, Max(kestus) AS pikim_magamine
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
WHERE kestus>INTERVAL '30' MINUTE
GROUP BY Magaja.magaja_id, eesnimi, perenimi
ORDER BY pikim_magamine;                                       (S)
```

Järgnevalt veel üks näide HAVING klausli (väär)kasutamise kohta. Ülesandeks on leida magajate arv, kellel on eesnimi määratud.

```
SELECT Count(*) AS arv
FROM Magaja
HAVING eesnimi IS NOT NULL;                                    (A)
```

See lause töötab küll MS Accessis, kuid ei lähe tööle andmebaasisüsteemides Oracle, PostgreSQL, Ocelot. SQL:1992 standardi järgi on selline päring illegaalne. SQL:1999 standard ütleb "Kui SELECT lause sisaldab HAVING klauslit, millele ei eelne GROUP BY klauslit, siis ei tohi SELECT klausel sisaldada ühtegi viidet FROM klauslis oleva tabeli veergudele, välja arvatud juhul, kui nendele veergudele rakendatakse kokkuvõttefunktsioon" (Gulutzan and Pelzer, 1999).

Küll aga töötab kõigis nimetatud andmebaasisüsteemides järgmine päring, mis leiab erinevate eesnimede arvu jättes arvestamata read, kus eesnimi on registreerimata.

```
SELECT eesnimi, Count(*) AS arv
FROM Magaja
GROUP BY eesnimi
HAVING eesnimi IS NOT NULL;                                    (APOS)
```

Siiski ei soovita eelnevalt toodud päringuid kasutada, sest suure tõenäosusega annab see andmebaasisüsteemile vihje, mitte kasutada päringu kiirendamiseks ühtegi indeksit.

Kõigepealt leiab andmebaasisüsteem read, mis vastavad WHERE klauslis toodud tingimustele (*ja vähendab seega grupeeritavate ridade arvu*), siis grupeerib ja alles kõige lõpuks rakendab HAVING klauslis toodud kitsendused. Seega parem lahendus sellele ülesandele oleks järgmine.

```
SELECT eesnimi, Count(*) AS arv
FROM Magaja
WHERE eesnimi IS NOT NULL
GROUP BY eesnimi;                                              (APOS)
```

HAVING klausel jäägu piirangute rakendamiseks kokkuvõttefunktsioonide tulemuste alusel.

Näide HAVING klausli kasutamise kohta ilma GROUP BY klauslita. Ülesandeks on väljastada magajate koguarv kui see on suurem kui 10. Kui magajate koguarv on 10 või vähem, siis ei tohi päring koguarvu väljastada.

```
SELECT Count(*) AS arv
FROM Magaja
HAVING Count(*)>10;                                            (APOS)
```

Mõnikord üritatakse kasutada järgnevat lahendust, sest *väidetavalt* on selle täitmine *mõnes* süsteemis kiirem võrreldes algse lahendusega, kus kasutatakse *Count* funktsiooni (igal juhul ei ole see reegel vaid sõltub konkreetsest süsteemist).

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Sum(1) AS magamiste_arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi
HAVING Sum(1)>4
ORDER BY Sum(1);                                               (APO)
```

Idee kohaselt liidetakse gruppi kuuluvate ridade arvu leidmiseks iga rea kohta tulemusse juurde väärtus 1. See lahendus kirjutab andmebaasisüsteemil ette loendamise protseduuri ja kokkuvõttes võib täitmise kiirus hoopis kannatada, sest see protseduur pole optimaalne ja andmebaasisüsteem oleks osanud valida *Count* funktsiooni kasutamise korral ise parema.

5. Ühendi leidmine

9. Esita päringu väljundis ühes veerus magajate perenimed ja asemete nimed. Pane teise veergu nime liigi kirjeldus: "Magaja perenimi" või "Aseme nimi". Sorteerida tulemus nimede järgi kasvavas järjekorras. Vihje: kasutage UNION relatsioonioperatsiooni ja konstanti SELECT klauslis.

```
SELECT  perenimi AS nimi, 'Magaja perenimi' AS liik
FROM Magaja
UNION SELECT nimi, 'Aseme nimi'
FROM Ase
ORDER BY nimi;                                          (APOS)
```

Nagu näete, võib SELECT klauslisse kirjutada konstante.

Veergude aliased peab määrama vaid esimeses SELECT lauses. Selleks, et määrata sorteerimistingimus üle kõikide SELECT lausete tulemuste, tuleb sorteerimistingimuses kasutada esimeses SELECT lauses määratud aliast.

Eelnev päring on samaväärne kui:

```
SELECT  perenimi AS nimi, 'Magaja perenimi' AS liik
FROM Magaja
UNION DISTINCT SELECT nimi, 'Aseme nimi'
FROM Ase
ORDER BY nimi;                                          (PS)
```

NB! UNION operatsioon kõrvaldab tulemusest mitmekordselt esinevad read, jättes alles vaid ühe, sest vaikimisi kasutatakse operaatorit UNION DISTINCT. Päringu tulemusest eemaldatakse korduvad read isegi siis, kui näiteks tabelis *Magaja* on kaks ühesugust perenime, aga tabelis *Ase* sellist nime ei ole.

Kui mitmekordselt esinevaid ridu ei soovita eemaldada, tuleb kasutada operaatorit UNION ALL.

```
SELECT perenimi
FROM Magaja
UNION ALL SELECT nimi
FROM Ase;                                               (APOS)
```

Olgu vaatluse all 3 SELECT päringut: A, B, C. Relatsioonialgebra operatsioon UNION on **assotsiatiivne**:

A UNION (B UNION C) on samaväärne kui (A UNION B) UNION C

Relatsioonialgebra operatsioon UNION on **kommutatiivne**: A UNION B = B UNION A

Paraku SQL peab oluliseks tabeli veergude järjekorda (igal veerul on oma järjekorranumber). See tähendab, et SQLi UNION (nagu ka INTERSECT) operatsioon rikub kommutatiivsuse põhimõtet (Date, 2006).

Oletame, et meil on kaks tabelit:

```
CREATE TABLE T1 (X INTEGER, Y INTEGER);
CREATE TABLE T2 (Y INTEGER, X INTEGER);
```

Päring 1	Päring 2
SELECT * FROM T1 UNION CORRESPONDING SELECT * FROM T2;	SELECT * FROM T2 UNION CORRESPONDING SELECT * FROM T1;

Kui lauses on määratud sõna CORRESPONDING, siis operatsioon tehakse *kõigi* veergude põhjal, mis on mõlemas tabelis ühesuguse *nimega.*

Päringu 1 tulemuseks on tabel, kus veerud on järjekorras (X, Y).
Päringu 2 tulemuseks on tabel, kus veerud on järjekorras (Y, X).

Kuna SQL arvestab tabelis veergude järjekorraga, siis SQLi jaoks ei ole tegemist kahe ühesuguse tabeliga ja seega ei kehti ka kommutatiivsuse printsiip!

```
SELECT  perenimi AS nimi, 'Magaja perenimi' AS liik
FROM Magaja
UNION CORRESPONDING SELECT nimi, 'Aseme nimi' AS liik
FROM Ase
ORDER BY nimi;                                                    (S)
```

NB! Järgneva päringus tulemusel toimub ühendi leidmine mitte samanimeliste veergude alusel, vaid veergude ***järjekorranumbrite*** alusel (esimene veerg esimese päringu tulemusest esimese veeruga teise päringu tulemuses; teine veerg esimese päringu tulemusest teise veeruga teise päringu tulemuses).

```
SELECT * FROM T1
UNION SELECT * FROM T2;
```

10. Tehke päring, millega leiate magaja identifikaatori, perenime ja aadressi. Kui aadressi ei ole registreeritud, tuleb selle asemel näidata telefoninumbrit. Kui nii aadressi kui ka telefoninumbrit ei ole registreeritud, siis peab väljas olema sõna "puudub".

Lahendus UNION operatsiooni kasutamisega.

```
SELECT  magaja_id, perenimi, aadress AS kontakt
FROM Magaja
WHERE aadress IS NOT NULL
UNION SELECT  magaja_id, perenimi, telefon
FROM Magaja
WHERE aadress IS NULL AND telefon IS NOT NULL
UNION SELECT  magaja_id, perenimi, 'puudub'
FROM Magaja
WHERE aadress IS NULL AND telefon IS NULL;                    (APO)
```

SQL standardit täielikult järgivas andmebaasisüsteemis võiks ülesande lahendamiseks kasutada ka *Coalesce* funktsiooni (tagastab vasakult lugedes esimese argumendi, mis ei ole NULL).

```
SELECT  magaja_id, perenimi, Coalesce(aadress, telefon,'puudub') AS
kontakt
FROM Magaja;                                                  (POS)
```

MS Accessis saab selle ülesande lahendamiseks kasutada ka süsteemi-defineeritud funktsiooni *Nz*.

```
SELECT magaja_id, perenimi,
Nz(aadress, Nz(telefon, 'puudub')) AS kontakt
FROM Magaja;                                                  (A)
```

Ühtlasi on see päring näide, kuidas saab funktsioonide väljakutseid üksteise sisse panna ja nii moodustada keerukamaid avaldisi.

Nz funktsioon: Nz (variant, [value_if_null])

Kui esimene argument on NULL, siis tagastab funktsioon teise argumendi väärtuse. Teine argument on valikuline. Kui esimene argument on NULL ja teine argument puudub, siis tagastab funktsioon NULL.

11. **Michalevicz et al. (2000, p. 364) defineerib funktsiooni, mille väärtus määrab inimese kuuluvuse *hägusasse* (ingl *fuzzy*) hulka "vanad inimesed":**
- **$m_A(X) = 0$, kui isik on 50 aastat vana või noorem**
- **$m_A(X) =$**

$$\left(1 + \frac{25}{(vanus\_aastates - 50)^2}\right)^{-1}$$

, kui isik on vanem kui 50 aastat. Funktsiooni väärtus 1 näitab, et isik kuulub *kindlasti* sellesse hulka ja väärtus 0 näitab, et isik *ei kuulu kindlasti* sellesse hulka. 0 ja 1 vahepealsed väärtused näitavad vahepealset kuuluvuse astet.

Teha päring, mis leiab selle funktsiooni väärtuse kõigi magajate jaoks, kellel on sünni aeg määratud. Tulemuses tuleb näidata magaja identifikaatorit, perenime, sünni aega ja funktsiooni $m_A(X)$ väärtust.

Allikas: Michalevicz, Z; Fogel, D, B. How to Solve It: Modern Heuristics. Springer, 2000. 467 p.

```
SELECT magaja_id, perenimi, 0 AS hinnang
FROM Magaja
WHERE ((date()-synni_aeg)/365.25)<=50
UNION SELECT magaja_id, perenimi, (1+(25/(((date()-synni_aeg)/365.25)-
50)^2))^-1 AS hinnang
FROM Magaja
WHERE ((date()-synni_aeg)/365.25)>50;                                    (A)
```

```
SELECT magaja_id, perenimi, 0 AS hinnang
FROM Magaja
WHERE ((CURRENT_DATE-synni_aeg)/365.25)<=50
UNION SELECT magaja_id, perenimi,  1/(1+25/(((CURRENT_DATE-
synni_aeg)/365.25-50)^2)) AS hinnang
FROM Magaja
WHERE ((CURRENT_DATE-synni_aeg)/365.25)>50;                              (P)
```

SQL standard näen ette CASE lause kasutamise võimaluse.

```
SELECT magaja_id, perenimi,
CASE
WHEN ((CURRENT_DATE-synni_aeg)/365.25)<=50 THEN 0
ELSE 1/(1+25/(((CURRENT_DATE-synni_aeg)/365.25-50)^2))
END AS hinnang
FROM Magaja;                                                             (P)
```

```
SELECT magaja_id, perenimi,
CASE
WHEN ((CURRENT_DATE-synni_aeg)/365.25)<=50 THEN 0
ELSE 1/(1+25/Power(((CURRENT_DATE-synni_aeg)/365.25-50),2))
END AS hinnang
FROM Magaja;                                                    (O)
```

12. **Leida tabelite *Magaja* ja *Ase* UNION JOIN. Valida tulemusse tabeli *Magaja* veerg perenimi ja tabeli *Ase* veerg *nimi*.**

<table expression 1>UNION JOIN<table expression 2>

MS Accessis (2013) UNION JOIN süntaksi kasutada ei saa. Soovitud tulemuse annab lause.

```
SELECT  perenimi AS magaja_perenimi, NULL AS aseme_nimi
FROM Magaja
UNION SELECT NULL, nimi
FROM Ase;                                                       (APO)
```

6. Alampäringud

13. Leia magajad, kelle perekonnanimi on sama, mis aseme nimi.

Tulemuse saamise jaoks võib kasutada alapäringut, mille tulemust kasutatakse põhipäringu tegemiseks: küsida asemete tabelist kõik nimed ja saadud tulemust kasutada põhipäringu WHERE-klauslis.

```
SELECT *
FROM Magaja
WHERE perenimi IN (SELECT nimi FROM Ase);                    (APOS)
```

Antud päringus kasutatav alampäring annab tulemuseks null või rohkem rida. Alampäring leiab asemete nimed. Iga magaja perenime puhul kontrollitakse, kas see kuulub asemete nimede hulka.

Ülesande lahendamiseks saab kasutada ka tabelite ühendamist (ingl *join*). Kahest tabelist valitakse ainult need read, mille vastavate veergude väärtused on võrdsed.

```
SELECT Magaja.*
FROM Magaja, Ase
WHERE Magaja.perenimi=Ase.nimi;                              (APOS)
```

Kui kirjutada eelmise SELECT lause SELECT klauslisse lihtsalt "*", siis väljastatakse andmed nii tabelist *Magaja* kui ka *Ase*.

NB! Tüüpiline viga. Alampäring võib tagastada rohkem kui ühe rea, aga kasutatakse lihtsat võrdluspredikaati.

```
SELECT *
FROM Magaja
WHERE perenimi = (SELECT nimi FROM Ase);
```

Õige oleks (ja seega veel üks võimalik lahendusvariant)

```
SELECT *
FROM Magaja
WHERE perenimi = ANY (SELECT nimi FROM Ase);                 (APOS)
```

või

```
SELECT *
FROM Magaja
WHERE perenimi = SOME (SELECT nimi FROM Ase);                (APOS)
```

14. Leia keskmisest pikemad asemed.

NB! Tüüpiline viga. Järgnev päring on vale ja ei tööta!!!

```
SELECT *
FROM Ase
WHERE pikkus>Avg(pikkus);
```

Õige oleks järgnev lahendus.

```
SELECT *
FROM Ase
WHERE pikkus>(SELECT Avg(pikkus) AS keskm FROM Ase);      (APOS)
```

Alampäring võib vastuseks anda ühe või mitu rida. Antud päringus kasutatav alampäring annab tulemuseks tabeli, kus on üks veerg (aste=1) ja null või üks rida (võimsus on 0 või 1).

Tegemist on skalaarse alampäringuga. Alampäring leiab asemete keskmise pikkuse. Keskmise arvutamisel arvestatakse vaid asemeid, mille pikkus on määratud. Saadud tulemust võrreldakse kõigi asemete pikkusega ja leitakse sellest pikemad asemed. Kui aseme pikkus on määramata, siis selliseid asemeid päring ei väljasta (sest tingimus *NULL>väärtus* kontroll annab tulemuseks *UNKNOWN*) ja neid ei võeta arvesse ka keskmise arvutamisel.

Selle ülesande võib ka lahendada järgnevalt. Skalaare alampäring on nüüd FROM klauslis ning selle moodustatava virtuaalse tabeli nimi on *keskmine*. Päringus moodustatakse selle alampäringu ja tabeli *Ase* otsekorrutis ning tulemuses esitatakse vaid read, kus pikkus tabelis *Ase* on suurem kui pikkus tabelis *keskmine.*

```
SELECT Ase.*
FROM Ase, (SELECT Avg(pikkus) AS keskm FROM Ase) AS keskmine
WHERE Ase.pikkus>keskmine.keskm;                          (AP)
```

```
SELECT Ase.*
FROM Ase, (SELECT Avg(pikkus) AS keskm FROM Ase) keskmine
WHERE Ase.pikkus>keskmine.keskm;                          (O)
```

15. Leia keskmisest rohkem kui 15 cm pikemad asemed.

```
SELECT *
FROM Ase
WHERE pikkus>(SELECT Avg(pikkus) + 15  AS keskm FROM Ase);
                                                          (APOS)
```

```
SELECT *
FROM Ase
WHERE pikkus>(SELECT Avg(pikkus) AS keskm FROM Ase)+15;
                                                          (APOS)
```

```
SELECT *
FROM Ase
WHERE pikkus-15>(SELECT Avg(pikkus) AS keskm FROM Ase);   (APOS)
```

Viimast varianti ei soovita praktikas kasutada, sest kui veerule *pikkus* oleks loodud indeks, siis veerule operaatori rakendamise tulemusel (pikkus-15) seda indeksit päringu täitmisel ei kasutata.

Alampäringu tulemuseks olevas tabelis võib veergudele määrata nime (WHERE pikkus-15>(SELECT Avg(pikkus) **AS keskm** FROM Ase)). SQL reeglite järgi pole see kohustuslik.

PostgreSQLis ja Oracles on võimalik päringus WITH klauslit kasutades defineerida nimeliselt üks või mitu alampäringut, millele saab edaspidi samas päringus viidata. Sellise võimaluse näeb ette ka hetkel kehtiv SQL standardi versioon (SQL:2011). Seega käesoleva ülesande lahendab ka järgnev lause.

```
WITH keskmisest_pikem AS (SELECT Avg(pikkus)+15 AS pikkus FROM
Ase)
SELECT *
FROM Ase
WHERE pikkus>(SELECT pikkus FROM keskmisest_pikem);          (PO)
```

16. Leia kõige vanemate magajate identifikaator, perenimi ja sünniaeg.

Lahendus 1.

```
SELECT magaja_id, perenimi, synni_aeg
FROM Magaja
WHERE synni_aeg=(SELECT Min(synni_aeg) AS minn FROM Magaja);

                                                          (APOS)
```

Lahendus 2.

```
SELECT magaja_id, perenimi, synni_aeg
FROM Magaja
WHERE synni_aeg NOT IN
(SELECT M1.synni_aeg
FROM Magaja M1, Magaja M2
WHERE M1.synni_aeg>M2.synni_aeg);

                                                          (APOS)
```

Alampäringus leitakse kõikvõimalikud sünniaegade paarid (otsekorrutis). Alampäringu tulemuseks on sellised sünniajad, millest leidub veel mõni varasem sünniaeg. Põhipäringuga leitakse sellise magaja andmed kelle sünniaeg ei sisaldu alampäringu leitud sünniaegade hulgas (st tema sünniaeg peab olema kõige varasem).

Lahendus *analüütilise funktsiooni* abil, mida saab kasutada PostgreSQL ja Oracle andmebaasis, leiab vanuse asetuse üldises vanuste pingereas ja seejärel leiab magajad, kelle vanus on pingereas esimene. Pingerida leitakse sünniaegade kasvavalt sorteerimise alusel, mis tähendab, et kõige väiksem sünniaeg (suurim vanus) on pingereas kõige esimene. Alampäring on antud juhul FROM, mitte WHERE klauslis.

```
SELECT magaja_id, perenimi, synni_aeg
FROM (SELECT magaja_id, perenimi, synni_aeg, rank() OVER (ORDER
BY synni_aeg ASC) AS asetus
FROM Magaja) foo
WHERE asetus=1;                                                    (PO)
```

17. Leia kõige vanemate ja kõige nooremate magajate identifikaator, perenimi ja sünniaeg.

```
SELECT magaja_id, perenimi, synni_aeg
FROM Magaja
WHERE synni_aeg IN (SELECT Min(synni_aeg) AS minn FROM Magaja)
UNION SELECT magaja_id, perenimi, synni_aeg
FROM Magaja
WHERE synni_aeg IN (SELECT Max(synni_aeg) AS minn FROM Magaja);
```

18. Millised magajad on sündinud hiljem, kui osteti kõige esimene ase?

```
SELECT *
FROM Magaja
WHERE synni_aeg>(SELECT Min(ostmise_kuupaev) AS minn FROM Ase);
```

Alampäringu tulemusena leitava tabeli veeru nimena ei saa ma kasutada *min*, sest see on SQLis reserveeritud sõna.

19. Leia magajate seast kõik naised, kes on vanemad kui vähemalt üks mees, kelle andmed on andmebaasi registreeritud.

```
SELECT *
FROM Magaja
WHERE sugu='N' AND synni_aeg<ANY(SELECT synni_aeg FROM Magaja
    WHERE sugu='M');
```

Alampäringuga leian kõige noorema mehe sünni aja. Seejärel leian andmed naiste kohta, kes on sellest mehest vanemad.

```
SELECT *
FROM Magaja
WHERE sugu='N' AND synni_aeg<(SELECT Max(synni_aeg) AS maks
    FROM Magaja WHERE sugu='M');
```

20. Leia magajate seast kõigist meestest vanemad naised.

```
SELECT *
FROM Magaja
WHERE sugu='N' AND synni_aeg<ALL(SELECT synni_aeg FROM Magaja
    WHERE sugu='M' AND synni_aeg IS NOT NULL);
```

Leitakse naissoost isikud, kelle sünniaeg on väiksem KÕIGIST alampäringu abil leitud (meeste) sünniaegadest.

Alampäringus peab olema kitsendus "synni_aeg IS NOT NULL". Kui seda ei oleks, tagastab alampäring NULL iga meessoost isiku kohta, kelle sünniaeg on määramata. Kuid sellisel juhul ei saa andmebaasisüsteem (ega ka inimkasutaja), kas põhipäringu leitud sünniaeg on väiksem KÕIGIST sünniaegadest (sest osa sünniaegu on teadmata) ja võrdluse tulemus on UNKNOWN. Lõpptulemusena ei tagasta päring ühtegi rida, kui alampäring tagastab vähemalt ühe NULLi.

Kui andmebaasis ei ole andmeid mitte ühegi mehe kohta (alampäring ei tagasta ühtegi rida), siis tagastab päring andmed kõikide naissoost magajate kohta.

21. Leia magajad, kes on vanemad, kui nendega samasoolised magajad keskmiselt.

```
SELECT *
FROM Magaja
WHERE synni_aeg< (SELECT Avg(synni_aeg) AS keskm
                  FROM Magaja AS Y
                  WHERE Y.sugu=Magaja.sugu);
```

Päringu koostamisel arvestan, et mida vanem inimene seda väiksem sünniaeg. Alampäringut täidetakse korduvalt, kui alampäring viitab pealausest pärit olevale veerule. Kui alampäring viitab pealausest pärit olevale veerule, on tegemist **korreleeruvate** päringutega. Need päringud on üksteisest sõltuvad päringud ja ei saa käivituda iseseisvalt). Igale välimise SELECT lause rea kohta käivitub (vähemalt kontseptuaalselt) eraldi alampäring.

Teine võimalus on koostada järgnev päring.

```
SELECT *
FROM Magaja
WHERE
(sugu='N' AND synni_aeg<(SELECT Avg(synni_aeg) FROM Magaja
WHERE sugu='N'))
OR
(sugu='M' AND synni_aeg<(SELECT Avg(synni_aeg) FROM Magaja
WHERE sugu='M')
);
```

Kuna veerus sugu on võimalik piiratud hulk väärtuseid – "M" või "N", siis on selline päring veel aktsepteeritav. Üldisemal juhul oleks muidugi kasulik kasutada esimest päringu varianti. Näiteks kui oleks ülesandeks leida kõik asemed mis on pikemad kui nendega samal aastal ostetud asemed keskmiselt, siis ei ole aastate kirjutamine WHERE klauslisse otstarbekas lahendus.

```
SELECT *
FROM Ase AS A1
WHERE pikkus>
(SELECT Avg(pikkus) FROM Ase AS A2
WHERE Year(A1.ostmise_kuupaev)=Year(A2.ostmise_kuupaev));
```

22. **Leia asemest nimega "Ase4" pikemad asemed. Leia see kahel viisil – kasutades alampäringut ja *self joini*.**

```
SELECT *
FROM Ase
WHERE pikkus>(SELECT pikkus FROM Ase WHERE nimi='Ase4');
```

See päring töötab vaid sellisel juhul, kui leidub vaid üks ase, mille nimi on "Ase4". Vastasel juhul tuleks alampäringu tingimuses kasutada ka aseme unikaalset identifikaatorit või kasutada põhipäringus koguselist võrdluse predikaati (>ALL). Antud juhul ei ole see vajalik, tänu andmebaasis jõustatud kitsendusele, mille kohaselt peavad tabeli *Ase* veerus *nimi* olema unikaalsed väärtused.

```
SELECT *
FROM Ase
WHERE pikkus>ALL(SELECT pikkus FROM Ase WHERE nimi='Ase4');
```

Teiseks võimaluseks on kasutada *self joini* ehk tabeli ühendamist iseendaga. Nüüd on tegemist otsekui kahe erineva tabeliga (virtuaalsed tabelid), millest esimest saame kasutada "Ase4" pikkuse leidmiseks ja teist sellest asemest pikemate asemete leidmiseks. Seda kõike tehakse ühe SELECT lause sees ning kõik vajalikud tingimused saab nüüd esitada selle SELECT lause WHERE- klauslis. *Self joini* puhul on kohustuslik eristada virtuaalsed tabelid. Selleks peab kasutama ühte või rohkemat aliast mille abil defineeritakse ühe tabeli põhjal nagu mitu erinevat muutujat (*range variable*) mille võimalikuks väärtuseks on tabelis olevad read.

```
SELECT A.*
FROM Ase AS A, Ase AS B
WHERE A.pikkus>B.pikkus
AND B.nimi='Ase4';
```

või

```
SELECT Ase.*
FROM Ase, Ase AS B
WHERE Ase.pikkus>B.pikkus
AND B.nimi='Ase4';
```

Kui tulemuses soovida päringut tulemuses näha ka aseme "Ase4" pikkust, siis võib teha päringu:

```
SELECT Ase.*, A4.pikkus AS aseme_Ase4_pikkus
FROM Ase, (SELECT pikkus FROM Ase WHERE nimi='Ase4') AS A4
WHERE Ase.pikkus>ALL(SELECT pikkus FROM Ase WHERE nimi='Ase4');
```

Antud juhul tekitatakse korrutis. Teine korrutises osalev tabel on virtuaalne tabel, mis sisaldab ainult aseme "Ase4" pikkust.

23. Leia võimalikult paljudel erinevatel viisidel selliste asemete nimed, millel on vähemalt üks kord magatud.

Üldistatult on tegu ülesandega, mis palub leida andmeid sellistest ühe tabeli ridadest, millel on vähemalt üks seotud rida teises tabelis. Relatsioonialgebras nimetatakse sellist operatsiooni poolühendamiseks (***semijoin***).

1) Lahendus mittekorreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE ase_id IN (SELECT ase_id FROM Magamine);
```

Alampäringuga leitakse tabelist *Ase* kõikide asemete identifikaatorid. Põhipäringuga leitakse nende asemete põhiandmed. Kuna alampäring võib leida rohkem kui ühe rea, siis kasutatakse IN predikaati.

2) Lahendus mittekorreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE ase_id=ANY (SELECT ase_id FROM Magamine);
```

2b) Lahendus mittekorreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE ase_id=SOME (SELECT ase_id FROM Magamine);
```

Kui aseme identifikaator tabelist *Ase* võrdub vähemalt ühe alampäringu poolt leitud aseme identifikaatoriga, siis on WHERE klauslis esitatud tingimus täidetud ja vastava aseme andmed kuuluvad päringu tulemusse. Sõnad ANY ja SOME on SQLis samaväärsed.

3) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE EXISTS
(SELECT magaja_id FROM Magamine WHERE
Magamine.ase_id=Ase.ase_id);
```

Viimases näites kasutatakse **korreleeruvat alampäringut**. Korreleeruva alampäringu korral nõuab alampäring infot peapäringu ridade kohta ja

käivitub iga kord uuesti iga peapäringu rea korral. Alampäring on korreleeruv peapäringuga, kui tema on viide peapäringus kasutatava tabeli veergudele.

Pärigu täitmine käib siin järgnevalt – iga aseme korral tabelist *Ase* täidetakse alampäring, mis otsib tabelist *Magamine* selliseid ridu, kus aseme identifikaator on võrdne peapäringu poolt vaadeldava aseme identifikaatoriga. Kui alampäring tagastab vähemalt ühe rea, siis on WHERE klauslis esitatud tingimus täidetud (EXISTS=TRUE) ja antud aseme andmed kuuluvad päringu tulemusse. Siis võetakse ette järgmine ase, täidetakse alampäring uuesti jne.

4) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE (SELECT Count(*) AS arv FROM Magamine WHERE
    Magamine.ase_id=Ase.ase_id)>0;
```

Leitakse asemed, millega seotud magamiste arv (ridade arv tabelis *Magamine*) on suurem kui 0.

5) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE ase_id =
(SELECT DISTINCT ase_id FROM Magamine WHERE
Ase.ase_id=Magamine.ase_id);
```

Korreleeruva alampäringuga üritatakse leida aseme identifikaator tabelis *Magamine.* Kui aseme identifikaatorit ei leita, siis tingimuse *ase_id=NULL* kontrolli tulemus on UNKNOWN ja vastavat aseme rida ei väljastata. Alampäringus tuleb kasutada DISTINCT, sest tabelis *Ase* võib olla mitu rida, kus on selline aseme identifikaator. Antud juhul võib kasutada võrdlusoperaatorit koos alampäringuga, sest tegu on korreleeruva alampäringuga mis võib tagastada maksimaalselt ühe rea.

6) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE ase_id IN
(SELECT DISTINCT ase_id FROM Magamine WHERE
Ase.ase_id=Magamine.ase_id);
```

7) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE EXISTS (SELECT Count(*) AS arv FROM Magamine WHERE
    Ase.ase_id=Magamine.ase_id HAVING Count(*)>0);
```

Alampäring tagastab vähemalt ühe rea (ja EXISTS predikaadi abil esitatud tingimus on täidetud), kui asemel toimunud magamiste arv on suurem kui 0. Ilma HAVING klauslita tagastaks alampäring iga magaja kohta tema magamiste arvu. HAVING klausli kasutamine antud kontekstis elimineerib

kõik sellised alampäringu tulemused, kus asemega seotud magamiste arv on 0. Kui magajal ei ole ühtegi seotud magamist (seotud magamiste arv on 0), siis ei tagasta alampäring ühtegi rida ja põhipäringu WHERE klauslis olev tingimus ei ole täidetud.

8) Lahendus tabelite ühendamisega.

```
SELECT DISTINCT Ase.nimi
FROM Magamine, Ase
WHERE Magamine.ase_id=Ase.ase_id;
```

Näidatakse ainult neid asemeid, millel on seotud andmeid magamiste tabelist. Ühel asemel võib olla magatud mitmel korral. Selleks, et iga aseme andmeid näidataks ainult üks kord, tuleb kasutada DISTINCT klauslit. Vastasel juhul näidatakse aseme andmeid nii mitu korda, kui temal on magatud.

8b) Tabelite ühendamise lause võib kirjutada ka teisiti.

```
SELECT DISTINCT Ase.nimi
FROM Ase INNER JOIN Magamine on Magamine.ase_id=Ase.ase_id;
```

NB! Antud lahendus erineb teistest selle poolest, et kui tabelis *Ase* on mitu ühesuguse nimega aset, siis antud lahendus näitab nime üks kord. Ilma DISTINCT kasutamiseta oleks tulemuses üks rida iga magamise kohta, kus ase on määratud.

Antud ülesande lahendused on ka kujukas näiteks selle kohta, et SQLis saab ühte probleemi väga mitmel erineval viisil lahendada. Sobivaima lahenduse valimise oluliseks kriteeriumiks on SQL lause täitmise kiirus (see võib erineda kordades).

24. Leia võimalikult paljudel erinevatel viisidel asemete nimed, millel pole ühtegi korda magatud.

Üldistatult on tegu ülesandega, mis palub leida andmeid sellistest ühe tabeli ridadest, millel pole ühtegi seotud rida teises tabelis. Relatsioonialgebras nimetatakse sellist operatsiooni poolvahe leidmiseks *(**semiminus*** või ***semidifference**)*. Kirjanduses kasutatakse ka mõnikord terminit ***anti-join***.

1)

```
SELECT nimi
FROM Ase
WHERE ase_id NOT IN (SELECT ase_id FROM Magamine  WHERE
ase_id IS NOT NULL);
```

2)

```
SELECT nimi
FROM Ase
WHERE ase_id<>ALL (SELECT ase_id FROM Magamine WHERE ase_id
IS NOT NULL);
```

Alampäringus leitakse kõigi asemete identifikaatorid, millel on magatud. Juhul, kui aseme identifikaator ei võrdu ühegi alampäringu poolt leitud

identifikaatoriga (<>ALL), siis on põhipäringu tingimus täidetud ja järelikult sellel asemel pole kordagi magatud.

3) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE NOT EXISTS
(SELECT * FROM Magamine WHERE Magamine.ase_id=Ase.ase_id);
```

4) Lahendus korreleeruva alampäringuga

```
SELECT nimi
FROM Ase
WHERE (SELECT Count(*) AS arv FROM Magamine WHERE
    Magamine.ase_id=Ase.ase_id)=0;
```

5) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE ase_id NOT IN (SELECT ase_id FROM Magamine  WHERE
Ase.ase_id=Magamine.ase_id);
```

6) Lahendus korreleeruva alampäringuga.

```
SELECT nimi
FROM Ase
WHERE EXISTS (SELECT Count(*) AS arv FROM Magamine WHERE
    Ase.ase_id=Magamine.ase_id HAVING Count(*)=0);
```

7)

```
SELECT Ase.nimi
FROM Ase LEFT JOIN Magamine ON Ase.ase_id = Magamine.ase_id
WHERE Magamine.ase_id IS NULL;
```

Kasutatakse ära *left join* operatsiooni omadust võtta kõik andmed ühest tabelist, sõltumata kas seotud tabelis on vastav rida olemas. Juhul, kui asemel ei ole ühtegi seotud magamist, tekib ühendamise tulemusse rida ka selle aseme kohta, kuid tabeli *Magamine* väljadele vastavad väljad on tühjad. Selle tingimuse täidetust kontrollides saabki leida asemed, millel pole seotud magamisi.

8) Lahendus tabelite ühendamise ja grupeerimisega.

```
SELECT nimi
FROM Ase LEFT JOIN Magamine ON Ase.ase_id = Magamine.ase_id
GROUP BY nimi
HAVING  Count(Magamine.ase_id)=0;
```

GROUP BY kasutamine eemaldab päringu tulemusest kordused, kuid antud juhul pole see probleem, sest andmebaasis jõustatud kitsenduse kohaselt on asemete nimed unikaalsed.

9) Lahendus, mis MS Accessis (2013) ei tööta, sest pole võimalik kasutada EXCEPT operaatorit.

```
SELECT nimi
FROM Ase
WHERE ase_id IN (
SELECT  ase_id FROM Ase
EXCEPT
SELECT ase_id FROM Magamine);
```

Alampäringus leitakse hulgateoreetilise vahe leidmise operatsiooni abil kõik asemed, millel ei ole magatud. Leitakse kaks asemete identifikaatorite hulka. Esimeses hulgas on kõikide asemete identifikaator ja teises on selliste asemete identifikaatorid, millel on vähemalt üks kord magatud. Alampäring leiab asemete identifikaatorid, mis on esimeses hulgas ja mida ei ole teises hulgas. Põhipäringuga leitakse asemete nimed.

10) Lahendus, mis MS Accessis (2013) ei tööta, sest pole võimalik kasutada EXCEPT operaatorit.

```
SELECT nimi
FROM Ase
EXCEPT
SELECT Ase.nimi FROM Ase INNER JOIN Magamine ON
Ase.ase_id=Magamine.ase_id;
```

Leiame kõikide asemete nimed ja seejärel nende asemete nimed, millel on vähemalt üks kord magatud ning lõpuks leiame nende hulkade vahe. Probleeme ei teki ka sellest, et selle päringu tulemusest eemaldatakse andmebaasisüsteemi poolt korduvad read automaatselt, sest asemete nimed on unikaalsed (andmebaasis on jõustatud vastav kitsendus).

25. Kui kaua on iga keskmisest vanem magaja kokku maganud? Tulemuses peab olema magaja identifikaator, perenimi ja tema magamiste kogukestus.

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Sum(kestus) AS
magamise_kogukestus_minutites
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
WHERE  Date()-synni_aeg>(SELECT Avg(Date()-synni_aeg) AS keskm
FROM Magaja)
GROUP BY Magaja.magaja_id, eesnimi, perenimi;
```

26. Kui kaua on keskmisest vanemad magajad kokku maganud? Tulemuses on ainult üks arv – magamiste kogukestus.

```
SELECT Sum(kestus) AS magamise_kogukestus_minutites
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
WHERE  Date()-synni_aeg>(SELECT Avg(Date()-synni_aeg) AS keskm
    FROM Magaja);
```

27. Leia magaja(d), kellega seoses on registreeritud kõige suurem arv magamisi. Näidata magaja identifikaatorit, eesnime, perenime ja tema magamiste arvu.

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Count(*) AS arv
FROM Magaja INNER JOIN Magamine ON
Magaja.magaja_id=Magamine.magaja_id
GROUP BY  Magaja.magaja_id, eesnimi, perenimi
HAVING Count(*)=(SELECT Max(arv) AS maks
                 FROM (SELECT Count(*) AS arv
                       FROM Magamine
                       GROUP BY magaja_id) AS ap);
```
(APOS)

Sisemine alampäring leiab erinevate magajate magamiste arvud (kellel on vähemalt üks seotud magamine). Väline alampäring leiab leiab nendest magamise arvudest maksimaalse. Kui ühelgi magajal ei ole seotud magamisi, siis ei leita päringu tulemusena ühtegi rida.

Teine võimalik lahendus:

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

Tingimusele vastavad magajad, kelle puhul põhipäringus leitud magamiste arv on suurem või võrdne kõigist alampäringuga leitud arvudest (suurima arvuga on see võrdne, kõigist teistest arvudest on see suurem).

MS Accessis, PostgreSQLis ja Oracles töötab ka järgnev päring, kus Max funktsiooni argumendiks olevad väärtused leitakse korreleeruva alampäringuga. Sisemise alampäringu abil leitakse erinevate magajate magamiste arv ning välise alampäringuga leitakse nende hulgast maksimaalne.

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

Ocelotis päringu täitmine ei õnnestu. SQL:1999 ei luba kokkuvõttefunktsiooni argumendina pöördumist alampäringu poole.

Kui tabelis *Magamine* ei ole ühtegi rida, siis ei ole eelnevalt esitatud päringute tulemuses ühtegi rida.

Ka järgnev päring lahendab selle ülesande. Erinevus eelnevatest lahendustest on, et kui tabelis *Magamine* ei ole ühtegi rida, siis tagastab päring kõigi magajate andmed. Milline tulemus on piirjuhul (tabelis *Magamine* pole ühtegi rida) kõige parem, sõltub konkreetsest olukorrast. Oluline on siinkohal meelde jätta, et SQL lausete testimisel tuleb pöörata tähelepanu ka piirjuhtudele (näiteks ühes või mitmes lauses kasutatavas tabelis ei ole ühtegi rida).

```
SELECT *
FROM (SELECT magaja_id, eesnimi, perenimi, (SELECT Count(*) AS arv
FROM Magamine WHERE Magamine.magaja_id=Magaja.magaja_id) AS
arv
FROM Magaja) magamiste_arv
WHERE arv>=ALL(SELECT (SELECT Count(*) AS arv
FROM Magamine WHERE Magamine.magaja_id=Magaja.magaja_id)
FROM Magaja);
```

(APO)

Sisemised alampäringud on korreleeruvad alampäringud, mille abil leitakse iga magajaga seotud magamiste arv.

28. Koosta päring, mis juhul, kui kasvõi üks ase on vaba (vaba=TRUE), tagastab teate "Veel on vabu asemeid". Vastasel juhul ei tohi päring mingeid andmeid tagastada (vihje – kasutage EXISTS predikaati).

```
SELECT DISTINCT 'Veel on vabu asemeid' AS Teade
FROM Ase
WHERE EXISTS (SELECT * FROM Ase WHERE vaba=TRUE);
```

EXISTS predikaati saab kasutada selleks, et kontrollida teatud ridade olemasolu ja määrata vastavalt põhipäringu tulemus.

Antud alampäringut täidetakse vaid ühel korral. Kui alampäring leiab vähemalt ühe rea, siis EXISTS predikaadi kontroll annab tulemuseks *TRUE.* Kui alampäring ei leia ühtegi rida siis EXISTS predikaadi kontroll annab tulemuseks *FALSE.*

Kui alampäring annab tulemuseks *TRUE* (st leidub veel mõni vaba ase), siis täidetakse põhipäring:

```
SELECT DISTINCT 'Veel on vabu asemeid' AS Teade
FROM Ase
```

DISTINCTi tuleb päringus kasutada, sest muidu oleks päringu tulemuseks niimitu ühesugust teadet, kuipalju on tabelis *Ase* ridu.

Kus EXISTS predikaati kasutada?

- Korreleeruvad alampäringud.
- Andmete esitusloogika realiseerimiseks. Näiteks meil on mingid andmed mida me soovime näidata alles siis, kui kõik read on ülevaadatud/töödeldud.

Ülesande lahendab ka päring:

```
SELECT DISTINCT 'Veel on vabu asemeid' AS Teade
FROM Ase
WHERE (SELECT COUNT(*) AS arv FROM Ase WHERE vaba=TRUE)>0
```

7. Kombineeritud päringud

29. Leidke sellised asemed, mille pikkus on üle 1.8 meetri. Päringu tingimuses tuleb kontrollida pikkust meetrites.

```
SELECT *
FROM Ase
WHERE pikkus/100>1.8;
```

Ülesande võib ka lahendada nii, et WHERE klauslis pole vaja taolist matemaatilist avaldist esitada.

```
SELECT ase_id, nimi, pikkus, laius, korgus, ostmise_kuupaev, eksisteerib,
vaba, kommentaar
FROM (SELECT Ase.*, pikkus/100 AS pikkus_meetrites
FROM Ase) AS ap
WHERE pikkus_meetrites>1.8;
```

ap on alampäringuga moodustatava virtuaalse tabeli nimi. SQL nõuab FROM klauslis alampäringule nime andmist isegi siis, kui sellele nimele lauses hiljem kordagi ei viidata.

30. Milline on magajate nimede (ees- ja perekonnanimi kokku) keskmine pikkus (märkide arv selles nimes)?

```
SELECT Avg(Len(eesnimi&perenimi)) AS keskmine_pikkus
FROM Magaja;
```

NB! Stringide ühendamiseks ei saa antud juhul kasutada operaatorit "+", sest string+NULL=NULL.

31. Leia, kui mitu protsenti magajate perenimedest algab "J" tähega.

```
SELECT ((SELECT Count(*) AS arv FROM Magaja WHERE perenimi LIKE
'J%')/Count(*)) * 100 AS protsent
FROM Magaja;
```

Päringu tulemus ei ole defineeritud, kui tabel *Magaja* on tühi – seal on null rida. Sellisel juhul tekib nulliga jagamine. Kui soovite sellist olukorda vältida võib kasutada *iif* funktsiooni. Selle funktsiooni esimese parameetri oodatav väärtus on tingimus. Teise parameetri oodatav väärtus on väärtus, mis tagastatakse, kui tingimus on täidetud. Kolmanda parameetri oodatav väärtus on väärtus, mis tagastatakse, kui tingimus ei ole täidetud.

```
SELECT ((SELECT Count(*) AS arv FROM Magaja WHERE perenimi LIKE
'J%')/iif(Count(*)=0,NULL,Count(*))) * 100 AS protsent
FROM Magaja;
```

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. Nulliga jagamisest tulenevat määramatust ei teki.

Ümardamiseks võib kasutada *Round* funktsiooni. Näiteks ümardamaks protsenti kahe kohani peale koma:

```
SELECT Round(((SELECT Count(*) AS arv FROM Magaja WHERE
perenimi LIKE 'J%')/iif(Count(*)=0,NULL,Count(*))) * 100, 2) AS protsent
FROM Magaja;
```

PostgreSQL ja Oracle andmebaasisüsteemide puhul tuleks *iif* funktsiooni asemel kasutada CASE avaldist. Ülesande lahendus Oracles.

```
SELECT Round((Count(CASE WHEN perenimi LIKE 'J%' THEN perenimi
ELSE NULL END)/ (CASE WHEN Count(*)=0 THEN NULL ELSE Count(*)
END))*100,2) protsent
FROM Magaja;
```

CASE avaldisega *CASE WHEN Count(*)=0 THEN NULL ELSE Count(*) END* tagatakse, et kui magajate arv on null, siis ei teki nulliga jagamist.

Ülesande lahendamiseks PostgreSQLis võib kasutada funktsiooni *Nullif,* millel on kaks parameetrit. Funktsioon tagastab NULL, kui parameetritele antud väärtused on võrdsed. Kui parameetritele antud väärtused ei ole võrdsed, siis tagastab funktsioon esimesele parameetrile antud väärtuse. ::decimal kasutatakse väärtuse teisendamiseks tüüpi *decimal*.

```
SELECT Round((Count(CASE WHEN perenimi LIKE 'J%' THEN perenimi
ELSE NULL END)::decimal/ Nullif(Count(*)::decimal,0))*100,2) AS protsent
FROM Magaja;
```

32. Leia magajate erinevate unikaalsete eesnimede arv.

SQL standard lubab järgnevat päringut.

```
SELECT Count(DISTINCT eesnimi) AS arv
FROM Magaja;
```

Selline päring andmebaasisüsteemis MS Access (2013) ei tööta. Üheks võimalikuks lahenduseks on teha päring kahes osas. Kõigepealt tuleb leida unikaalsed eesnimed.

```
SELECT DISTINCT eesnimi
FROM Magaja;
```

Salvestame päringu nime all: *Unikaalsed_eesnimed*. Seejärel tuleb teha päring:

```
SELECT Count(eesnimi) AS arv
FROM Unikaalsed_eesnimed;
```

SQL standard lubab (alates SQL:1992) FROM klauslis tabeli nime asemel kasutada alampäringut. Seega on võimalik teha ka päring:

```
SELECT Count(eesnimi) AS arv
FROM (SELECT DISTINCT eesnimi FROM Magaja) AS ap;
```

Kui *Count* funktsiooni argumendiks on veeru nimi, siis loetakse kokku read, kus antud veerus on väärtus määratud (ei ole NULL). Alternatiivsed päringud kasutavad otsingutingimust, et jätta vaatluse alt välja read, kus eesnime väärtus puudub

```
SELECT Count(*) AS arv
FROM (SELECT DISTINCT eesnimi FROM Magaja WHERE eesnimi IS
    NOT NULL) AS ap;
```

```
SELECT Count(*) AS arv
FROM (SELECT DISTINCT eesnimi FROM Magaja) AS ap
WHERE eesnimi IS NOT NULL;
```

ap – alampäringu tulemusena moodustatava virtuaalse tabeli nimi. MS Accessis võib selle nime ka jätta määramata, kuid siis annab MS Access ise selle nime. Nimi võib olla näiteks selline: [%$##@_Alias]

NB! Tüüpiline viga. Järgnev päring leiab isikute arvu, kellel on eesnimi määratud. See päring ei leia *erinevate* eesnimede arvu. Kõigepealt leitakse isikute arv (üks arv). Sellele rakendatakse DISTINCT (korduste kõrvaldamine) ja alles jääb ikka see sama arv.

```
SELECT DISTINCT Count(eesnimi) AS arv
FROM Magaja;
```

33. Leia magajad, kelle eesnimi sisaldub perenimes (näide: John Johnson).

```
SELECT *
FROM Magaja
WHERE perenimi LIKE '%' & eesnimi & '%';
```

Juhul, kui andmebaasisüsteem eristab stringikonstantides suuri ja väikeseid tähti (vaikimisi käitumine), leiab eelnev päring Celko (1999) andmeil nime "***Taavi Taavi***ste", kuid mitte nime "***Ain Vain***", sest päringu tingimuses eristatakse suuri ja väikeseid tähti. Selle vältimiseks võib teha päringu.

```
SELECT *
FROM Magaja
WHERE Upper(perenimi) LIKE '%' & Upper(eesnimi) & '%';
```

NB! MS Accessis stringikonstantides suuri ja väikesi tähti ei eristata.

34. **Leia tabeli *Magaja* põhjal järgmised andmed.**
- **Magaja sünniaeg formaadis YYYY-MM-DD.**
- **Magaja sündimise päev (arv).**
- **Magaja sündimise kuu (arv).**
- **Magaja sündimise kuu (nimi).**
- **Magaja sündimise aasta (arv).**
- **Eesnime esimene täht.**
- **Perenime tähed 3-5.**
- **Perenime pikkus.**
- **Magaja aadress. Kui aadress on määramata, näidata selle asemel fraasi "Aadressi ei ole".**

Vihje: Vajalikud funktsioonid: Format, Day, Month, MonthName, iif, IsNull, Year, Left, Mid, Nz.

MS Accessi süsteemi-defineeritud funktsioonide kohta saab näiteks infot: http://www.techonthenet.com/access/functions/

```
SELECT
Format(synni_aeg,'YYYY-MM-DD') AS sünniaeg,
Year(synni_aeg) AS sünniaasta,
Month(synni_aeg) AS sünnikuu,
iif(IsNull(synni_aeg), null, MonthName(Month(synni_aeg))) AS
sünnikuu_sonadega,
Day(synni_aeg) AS sünni_päev,
Left(eesnimi,1) AS eesnime_esitäht,
Mid(perenimi,3,2) AS perenime_tykk,
Nz(aadress,'Aadressi ei ole') AS aadr
FROM Magaja;
```

Avaldis: *iif(IsNull(synni_aeg), null, MonthName(Month(synni_aeg)))* esitab sellise tingimuslause:

```
IF synni_aeg IS NULL THEN
 {NULL }
ELSE {
 MonthName(Month(synni_aeg))
}
```

Funktsioonide *Year*, *Month* ja *Day* asemel võib kasutada ka funktsiooni *DatePart.* Näiteks sünniaasta leidmiseks võib kasutada ka avaldist: *DatePart('yyyy',synni_aeg) AS sünniaasta*.

35. **Mitu protsenti moodustab kõige vanema magaja vanus kõigi magajate vanuste summast? Arvutuse tegemisel kasutage vanust päevades. Ümarda protsent täisarvuks.**

```
SELECT Round((Max(Date()-synni_aeg)*100)/(Sum(Date()-synni_aeg)),0)
AS protsent
FROM Magaja;
```

Tegemist on lihtsa protsendiarvutusega:

Summaarne vanus	100%
Vanima magaja vanus	X%

X= (Vanima magaja vanus*100)/Summaarne vanus

36. Leia iga magaja 2001 aastal alanud magamiste arv. Väljasta magaja identifikaator, eesnimi, perenimi ja magamiste arv. Kui sellel aastal pole andmeid ühegi magamise algamise kohta, siis magamiste arv=0.

Vale lahendus:

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Count(Maga.magaja_id) AS
arv
FROM Magaja LEFT JOIN Magamine  AS Maga ON
Magaja.magaja_id=Maga.magaja_id
WHERE Year(algus)=2001
GROUP BY Magaja.magaja_id, eesnimi, perenimi;
```

Outer-join operatsiooni tulemuses on andmed ka selliste magajate kohta, kellel pole seotud magamisi. Ühendamise operatsiooni tulemusena leitud ridadele rakendatakse WHERE klauslis esitatud kitsendus. Selle tulemusel eemaldatakse vaatluse alt read, kus magamise algus <>2001 või kus magamise algus on määramata. Lõppkokkuvõttes ei ole päringu tulemuses andmeid magajate kohta, kellel ei ole ühtegi seotud magamist (mis on viga).

Lahendus 1

```
SELECT Magaja.magaja_id, eesnimi, perenimi,
Count(Magamine.magaja_id) AS arv
FROM Magaja LEFT JOIN (SELECT magaja_id FROM Magamine WHERE
Year(algus)=2001) AS Maga ON Magaja.magaja_id=Maga.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi;
```

Tabelis *Magaja* olevad read ühendatakse selliste ridadega tabelis *Magamine*, kus algus=2001. Päringu tulemuses on andmed ka magajate kohta, kellel selliseid seotud magamisi ei ole.

Lahendus 2

```
SELECT Magaja.magaja_id, eesnimi, perenimi, (SELECT Count(*) AS arv
FROM Magamine WHERE Magaja.magaja_id=Magamine.magaja_id AND
Year(algus)=2001) AS arv
FROM Magaja;
```

Lahendus 3

```
SELECT Magaja.magaja_id, eesnimi, perenimi,
Count(Magamine.magaja_id) AS arv
FROM Magaja LEFT JOIN Magamine ON
(Magaja.magaja_id=Magamine.magaja_id AND Year(algus)=2001)
GROUP BY Magaja.magaja_id, eesnimi, perenimi;
```

37. Millistel asemetel on maganud kokku kõige kauem maganud magaja? Sorteerige asemed nime järgi.

Ülesande osad.
a. Leia kõigi magajate magamiste kestvus.
b. Leia kokku kõige kauem maganud magaja.
c. Leia asemed, millel see magaja on maganud ja sorteeri need laiuse järgi.

Päring 1. Leiab kõigi kunagi maganud magajate magamiste kogukestused.

```
SELECT Magaja.magaja_id, eesnimi, perenimi, Sum(kestus) AS kogukestus
FROM Magaja INNER JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Magaja.magaja_id, eesnimi, perenimi;
```

Salvestan ja paneme nimeks: *Magamiste_kestus.*

Päring 2. Leiab salvestatud päringu (virtuaalse tabeli) *Magamiste_kestus* põhjal kokku kõige rohkem maganud magaja.

```
SELECT *
FROM Magamiste_kestus
WHERE kogukestus=(SELECT Max(kogukestus) AS maks FROM
Magamiste_kestus);
```

Salvestan ja paneme nimeks: *Suurim_unimüts.*

Päring 3. Leiab asemed, millel on maganud kõige rohkem maganud magaja.

```
SELECT DISTINCT Ase.ase_id, Ase.nimi AS aseme_nimi, Ase.laius AS
aseme_laius, Ase.pikkus AS aseme_pikkus, Suurim_unimüts.eesnimi,
Suurim_unimüts.perenimi
FROM Suurim_unimüts INNER JOIN (Ase INNER JOIN Magamine ON
Ase.ase_id = Magamine.ase_id) ON Suurim_unimüts.magaja_id =
Magamine.magaja_id
ORDER BY Ase.nimi;
```

Salvestan nime all: *Suurima_unimütsi_lemmikasemed.*

DISTINCT läheb vaja, et mitte väljastada asemeid mitmekordselt.

Teisel juhul tulevad kõigepealt read, kus on pikkus ja kõige lõpus read, kus pikkus pole määratud.

Lahendus ühe SQL lausega:

```
SELECT DISTINCT Ase.ase_id, Ase.nimi AS aseme_nimi, Ase.laius AS
    aseme_laius, Ase.pikkus AS aseme_pikkus, Magaja.eesnimi,
    Magaja.perenimi
FROM ((Magaja INNER JOIN (SELECT magaja_id
FROM Magamine
GROUP BY magaja_id
HAVING Sum(kestus)=(SELECT Max(kogukestus) AS maks FROM
(SELECT Sum(kestus) AS kogukestus FROM Magamine GROUP BY
    magaja_id))) AS Suurim_unimüts ON
    Magaja.magaja_id=Suurim_unimüts.magaja_id) INNER JOIN
Magamine ON Suurim_unimüts.magaja_id=Magamine.magaja_id) INNER
    JOIN Ase ON Magamine.ase_id=Ase.ase_id
ORDER BY Ase.nimi;
```

NB! Ilmtingimata ei ole vaja esimese korraga ülesanne ühe SQL lausega ära lahendada. Proovige ülesannet lahendada vaadete kasutamise abil. Mitme SQL lause üheks kokku panemine on hiljem juba lihtsam. Vahetulemuste vaadetena vormistamisel on ka see hea külg, et neid saab hiljem taaskasutada teiste päringute juures.

38. Milline on kõigi nende asemete pindala, kus on maganud kõige kauem maganud magaja? Tulemuses peab iga ase olema ainult üks kord.

Ülesande osad:
1 Eelmise ülesande abil on leitud vajalik asemete hulk.
2 Arvuta iga aseme pindala (pikkus*laius).

```
SELECT aseme_laius*aseme_pikkus AS aseme_pindala, aseme_nimi
FROM Suurima_unimütsi_lemmikasemed
GROUP BY aseme_laius*aseme_pikkus, aseme_nimi
ORDER BY aseme_laius*aseme_pikkus;
```

NB! Alates SQL:1999 on ORDER BY klauslis lubatud kasutada ka avaldisi:

```
SELECT *
FROM  Ase
ORDER BY pikkus * (-1);
```

annab erineva tulemuse kui

```
SELECT *
FROM Ase
ORDER BY pikkus DESC;
```

Erinevus tuleb NULLide käsitlemisest. Esimesel juhul tulevad kõigepealt read, kus pole pikkust määratud ja seejärel read, kus on pikkus määratud. Teisel juhul tulevad MS Accessis kõigepealt read, kus on pikkus määratud ja siis read, kus pikkus ei ole määratud.

39. Milline on kõigi nende asemete kogupindala, kus on maganud kõige kauem maganud magaja? Kogupindala arvutamise võtke iga vaadeldava ase arvesse ainult üks kord.

Ülesande osad.
- Eelmise ülesande abil on leitud vajalike asemete pindalad.
- Arvuta asemete kogupindala.

```
SELECT Sum(aseme_laius*aseme_pikkus) AS aseme_pindala
FROM Suurima_unimütsi_lemmikasemed;
```

40. Mitu erinevat magajat on maganud aseme(te)l, mille(de)l on kokku kõige kauem magatud?

```
SELECT Count(*) AS arv
FROM
(SELECT DISTINCT ase_id, magaja_id
FROM Magamine
WHERE ase_id IN
(SELECT ase_id
FROM  Magamine
GROUP BY ase_id
HAVING Sum(kestus)=(SELECT Max(summa) AS maks FROM
(SELECT Sum(kestus) AS summa FROM Magamine GROUP BY ase_id)
AS summad))) AS ap;
```

Alampäring	***Selgitus***
SELECT Sum(kestus) AS summa FROM Magamine GROUP BY ase_id	Leitakse igal asemel toimunud magamise kogukestus
SELECT Max(summa) AS maks FROM (SELECT Sum(kestus) AS summa FROM Magamine GROUP BY ase_id) AS summad	Leitakse maksimaalne magamise kogukestus, mis mõnel asemel on toimunud.
SELECT ase_id FROM Magamine GROUP BY ase_id HAVING Sum(kestus)=(SELECT Max(summa) AS maks FROM (SELECT Sum(kestus) AS summa FROM Magamine GROUP BY ase_id) AS summad)	Leitakse ase(med), kus on kokku kõige rohkem magatud.
SELECT DISTINCT ase_id, magaja_id FROM Magamine WHERE ase_id IN (SELECT ase_id	Leitakse magajad (ühekordselt), kes on leitud asemetel maganud.

Alampäring	*Selgitus*
FROM Magamine GROUP BY ase_id HAVING Sum(kestus)=(SELECT Max(summa) AS maks FROM (SELECT Sum(kestus) AS summa FROM Magamine GROUP BY ase_id) AS summad))	
SELECT Count(*) AS arv FROM (SELECT DISTINCT ase_id, magaja_id FROM Magamine WHERE ase_id IN (SELECT ase_id FROM Magamine GROUP BY ase_id HAVING Sum(kestus)=(SELECT Max(summa) AS maks FROM (SELECT Sum(kestus) AS summa FROM Magamine GROUP BY ase_id) AS summad))) AS ap;	Loetakse magajate arv kokku.

Teine võimalik lahendus:

```
SELECT Count(*) AS arv
FROM
(SELECT DISTINCT ase_id, magaja_id
FROM Magamine
WHERE ase_id IN
(SELECT ase_id
FROM  Magamine
GROUP BY ase_id
HAVING Sum(kestus)>=ALL(SELECT Sum(kestus) AS summa FROM
Magamine GROUP BY ase_id)));
```

Tingimusel vasta kestuste summa peab olema suurem või võrdne kõigi alampäringuga leitud summadega. Tingimusele vastab suurim kestuse summa, sest ühe alampäringu leitud summaga on see võrdne ja teistest on see suurem.

41. Kui palju kordi on igal asemel magatud? Näita tulemuses aseme nime ja sellel magamiste arvu.

```
SELECT nimi, Count(Magamine.ase_id) AS magamiste_arv
FROM  Ase LEFT JOIN Magamine ON Ase.ase_id=Magamine.ase_id
GROUP BY Ase.ase_id, nimi;
```

Count funktsiooni argumendina tuleb kasutada praegu veeru nime. Miks? Kasutatakse välisühendamist, et näha tulemuses ka asemeid, kus pole kordagi magatud. Tulemuses on vähemalt üks rida iga aseme kohta (ka siis, kui sellel asemel pole kordagi magatud). Kui kasutada *Count(*)*, siis asemetel, millega pole seotud magamisi, loetakse magamiste arvuks 1. *Count* funktsioon tuleb rakendada mingile tabelis *Magamine* olevale veerule, et võtta ühendamise tulemuse põhjal magamiste arvu määramisel arvesse vaid need read, kus selles veerus on väärtus olemas.

SQL standard lubab skalaarset (ühe rea ja veeru tagastavat) SELECT lauset kasutada SELECT lauses konstandi asemel. Seega sama tulemuse annab ka päring

```
SELECT nimi, (SELECT Count(*) AS arv FROM Magamine WHERE
Magamine.ase_id=ase.ase_id) AS magamiste_arv
FROM Ase;
```

42. Leia magajad, kes ei ole maganud asemel, mille identifikaator on 3. Väljasta päringu tulemuses magaja identifikaator, eesnimi ja perenimi.

Lahendus 1:

```
SELECT magaja_id, eesnimi, perenimi
FROM Magaja
WHERE magaja_id NOT IN (SELECT magaja_id FROM Magamine WHERE
ase_id=3);
```

Alampäringuga leitakse magajad, kes on vähemalt üks kord maganud asemel, mille identifikaator on 3. Põhipäringuga leitakse magajad, kelle identifikaator *ei kuulu* alampäringuga leitud identifikaatorite hulka.

Lahendus 2:

```
SELECT Magaja.magaja_id, Magaja.eesnimi, Magaja.perenimi
FROM Magaja LEFT JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Magaja.magaja_id, Magaja.eesnimi, Magaja.perenimi
HAVING Max(iif(Magamine.ase_id=3,1,0))=0;
```

iif funktsiooni poole pöördumine esitab praegu avaldise:

```
IF Magamine.ase_id=3 THEN
    {1}
ELSE
    {0}
END IF;
```

iif funktsiooni asemel saab kasutada paljudes teistes andmebaasisüsteemides CASE avaldist (mis on ette nähtud ka SQL standardis).

43. Leia selliste magajate identifikaatorid, kes on maganud ainult asemel, mille identifikaator on 3 ning mitte ühelgi teisel asemel.

```
SELECT DISTINCT Magamine.magaja_id
FROM (SELECT ap.magaja_id
FROM (SELECT DISTINCT Magamine.ase_id, Magamine.magaja_id
FROM Magamine) AS ap
GROUP BY ap.magaja_id
HAVING Count(*)=1) AS yhel_asemel_maganud INNER JOIN Magamine
ON yhel_asemel_maganud.magaja_id=Magamine.magaja_id
WHERE Magamine.ase_id=3;
```

Sinisega esitatud alampäringuga leitakse selliste magajate identifikaatorid, kes on maganud *ainult ühel asemel*. DISTINCT klausli kasutamine alampäringus tagab, et ei arvestata kui mitu korda on magaja sellel asemel maganud. Alampäringu tulemus ühendatakse tabeliga *Magamine*, et kontrollida, kas magaja on maganud asemel identifikaatoriga 3.

```
SELECT DISTINCT magaja_id
FROM Magamine
WHERE magaja_id NOT IN (SELECT magaja_id FROM Magamine WHERE
ase_id<>3);
```

Alampäringuga leitakse magajad, kes on kunagi maganud asemel, mille identifikaator *ei ole* 3. Põhipäringuga leitakse sellised magajad, kelle kohta on registreeritud vähemalt üks magamine, kuid kelle identifikaator ei sisaldu alampäringuga leitud identifikaatorite hulgas. Järelikult saavad nad olla maganud ainult asemel identifikaatoriga 3.

44. Leia selliste magajate identifikaatorid, kes on maganud nii asemel, mille identifikaator on 1 kui ka asemel, mille identifikaator on 2.

Lahendus 1. Alampäringutega leitakse magajate identifikaatorid, kes on maganud asemetel 1 ja 2. Põhipäringus kontrollitakse, kas magaja identifikaator, mis on leitud tabelist *Magaja*, sisaldub alampäringuga leitud identifikaatorite hulgas.

```
SELECT magaja_id
FROM Magaja
WHERE magaja_id IN (SELECT magaja_id FROM Magamine WHERE
ase_id=1)
AND magaja_id IN (SELECT magaja_id FROM Magamine WHERE
ase_id=2);
```

Lahendus 2. Kasutab korreleeruvaid alampäringuid. Näiteks tingimusega *1 IN (SELECT ase_id FROM Magamine WHERE Magaja.magaja_id= Magamine.magaja_id)* kontrollitakse, kas ase identifikaatoriga 1 sisaldub nende asemete hulgas, millel konkreetne magaja on maganud.

```
SELECT magaja_id
FROM Magaja
WHERE 1 IN (SELECT ase_id FROM Magamine WHERE
Magaja.magaja_id=Magamine.magaja_id)
AND 2 IN (SELECT ase_id FROM Magamine WHERE
Magaja.magaja_id=Magamine.magaja_id);
```

Lahendus 3. Kasutab korreleeruvaid alampäringuid. Näiteks tingimusega *EXISTS (SELECT * FROM Magamine WHERE Magaja.magaja_id= Magamine.magaja_id AND Magamine.ase_id=1)* kontrollitakse, kas konkreetse magajaga on seotud asemel 1 toimunud magamisi.

```
SELECT magaja_id
FROM Magaja
WHERE EXISTS (SELECT * FROM Magamine WHERE
Magaja.magaja_id=Magamine.magaja_id AND Magamine.ase_id=1)
AND EXISTS (SELECT * FROM Magamine WHERE
Magaja.magaja_id=Magamine.magaja_id AND Magamine.ase_id=2);
```

Lahendus 4.

```
SELECT magaja_id
FROM Magamine
WHERE ase_id IN (1,2)
GROUP BY magaja_id
HAVING Min(ase_id)<>Max(ase_id);
```

Leian asemetel identifikaatoriga 1 või 2 maganud magajad (SELECT magaja_id FROM Magamine WHERE ase_id IN (1,2)). Grupeerimise ja HAVING klausliga kontrollitakse, et see magaja on maganud mõlemal asemel.

Lahendus 5.

```
SELECT DISTINCT M1.magaja_id
FROM Magamine AS M1 INNER JOIN Magamine AS M2 ON
M1.magaja_id=M2.magaja_id
WHERE M1.ase_id=1 AND M2.ase_id=2;
```

Lahendus kasutab tabeli ühendamist iseendaga (self-join).

45. Leia selliste magajate identifikaatorid, kes on kindlasti maganud asemel, mille identifikaator on 1 ja võibolla on maganud ka asemel, mille identifikaator on 2 (st neil on 0 või rohkem magamist asemel identifikaatoriga 2).

Me saame ülesannet lihtsustada (viia samaväärsele kuid lihtsamale kujule) jättes tähelepanuta ülesande teise poole (*ja võibolla on maganud ka asemel, mille identifikaator on 2 (st neil on 0 või rohkem magamist asemel identifikaatoriga 2)*).

```
SELECT DISTINCT magaja_id
FROM Magamine
WHERE ase_id=1;
```

46. Leia selliste magajate identifikaatorid, kes on kas maganud asemel identifikaatoriga 1 või asemel identifikaatoriga 2, kuid mitte mõlemal.

Lahendus 1.

```
SELECT magaja_id
FROM (SELECT DISTINCT magaja_id, ase_id FROM Magamine) AS ap
GROUP BY magaja_id
HAVING Sum(iif(Magamine.ase_id IN (1,2),1,0))=1;
```

Alampäringuga leitakse *magaja_id* ja *ase_id* paarid, kus vastava identifikaatoriga magaja on maganud vastava identifikaatoriga asemel. Päringu tulemusest on eemaldatud korduvad read. *iif* funktsiooni poole pöördumine esitab praegu avaldise:

```
IF Magamine.ase_id IN (1,2) THEN
    {1}
ELSE
    {0}
END IF;
```

Kui magaja on maganud nii asemel 1 kui ka asemel 2, siis on *Sum(iif(Magamine.ase_id IN (1,2),1,0))* tulemus 2.

Kui magaja ei ole maganud ei asemel 1 ega ka asemel 2, siis on *Sum(iif(Magamine.ase_id IN (1,2),1,0))* tulemus 0.

Lahendus 2.

```
SELECT DISTINCT magaja_id
FROM Magamine
WHERE ase_id IN (1,2) AND magaja_id NOT IN (SELECT M1.magaja_id
FROM Magamine AS M1 INNER JOIN Magamine AS M2 ON
M1.magaja_id=M2.magaja_id WHERE M1.ase_id=1 AND M2.ase_id=2);
```

Alampäringus kasutatakse *self joini* ning sellega leitakse magajad kes on maganud nii asemel identifikaatoriga 1 kui ka asemel identifikaatoriga 2. Põhipäringuga leitakse magajad kes on samuti maganud kas asemel

identifikaatoriga 1 või asemel identifikaatoriga 2 kuid kelle identifikaator ei sisaldu alampäringu tulemuses. Järelikult selline magaja on maganud kas asemel identifikaatoriga 1 või asemel identifikaatoriga 2 aga mitte mõlemal korraga.

Lahendus 3. MS Accessis saab kasutada loogikaoperaatorit XOR – välistav või.

```
SELECT magaja_id
FROM Magaja
WHERE magaja_id IN (SELECT magaja_id FROM Magamine WHERE
ase_id=1)
XOR magaja_id IN (SELECT magaja_id FROM Magamine WHERE
ase_id=2);
```

47. **Leia magajad, kes on maganud kahel või vähemal *erineval* asemel. Väljasta päringu tulemuses magaja identifikaator, eesnimi ja perenimi.**

Lahendus 1.

```
SELECT Magaja.magaja_id, Magaja.eesnimi, Magaja.perenimi
FROM Magaja LEFT JOIN (SELECT DISTINCT magaja_id, ase_id FROM
Magamine) AS MM ON Magaja.magaja_id = MM.magaja_id
GROUP BY Magaja.magaja_id, Magaja.eesnimi, Magaja.perenimi
HAVING Count(MM.magaja_id)<=2;
```

Sinisega esitatud alampäringuga leitakse sellised *magaja_id* ja *ase_id* paarid, mille korral on vastava identifikaatoriga magaja maganud vastava identifikaatoriga asemel. Alampäringu tulemusest on korduvad read eemaldatud. Sellist alampäringut on vaja, et arvestada ka magajatega, kellel on rohkem kui kaks seotud magamist, kuid mis on toimunud kahel või vähemal erineval asemel.

Päringus tuleb kasutada välisühendamist, et leida ka magajad, kellel ei ole ühtegi seotud magamist. *Count* funktsiooni argumendiks on veeru identifikaator (Magamine.magaja_id), et magajate puhul, kellel pole seotud magamisi, oleks *Count* funktsiooni poole pöördumise tulemus 0.

Lahendus 2.

```
SELECT magaja_id, eesnimi, perenimi
FROM Magaja
WHERE magaja_id  NOT IN
(SELECT M1.magaja_id
FROM Magamine AS M1, Magamine AS M2, Magamine AS M3
WHERE M1.magaja_id=M2.magaja_id AND M2.magaja_id=M3.magaja_id
AND M1.ase_id<M2.ase_id AND M2.ase_id<M3.ase_id );
```

See lahendus kasutab alampäringus *self joini* (tabeli ühendamist iseendaga). Alampäringuga leitakse sellised magajad, kes on maganud rohkem kui kahel (st vähemalt kolmel) erineval asemel. Põhipäringuga leitakse magajad kelle identifikaator ei sisaldu alampäringuga leitud identifikaatorite hulgas.

48. Leia, kui mitu protsenti moodustab igal asemel magamiste arv kõigist asemetel magamistest. Näita tulemuses aseme identifikaatorit, nime ja protsenti. Näidake tulemuses kõiki asemeid – kui asemel pole magatud, siis on sellel magamiste protsent 0.

Lahendus 1.

```
SELECT Ase.ase_id, nimi, Round((Count(Magamine.ase_id)*100)/(SELECT
Count(*) AS arv FROM Magamine),2) AS magamiste_protsent
FROM Ase LEFT JOIN Magamine ON Ase.ase_id=Magamine.ase_id
GROUP BY Ase.ase_id, nimi;
```

Lahendus 2.

```
SELECT Ase.ase_id, nimi, Round(((SELECT Count(*) AS arv FROM
Magamine WHERE Magamine.ase_id=Ase.ase_id)*100)/(SELECT Count(*)
AS arv FROM Magamine),2) AS magamiste_protsent
FROM Ase;
```

Korreleeruv alampäring *(SELECT Count(*) AS arv FROM Magamine WHERE Magamine.ase_id=Ase.ase_id)* leiab iga aseme kohta sellel magamiste arvu. Alampäring *(SELECT Count(*) AS arv FROM Magamine)* leiab magamiste koguarvu.

Vaadake ülesande 31 lahenduse kirjeldusest, kuidas vältida nulliga jagamist.

49. Leia asemete nimed, millel on 2000 aastal vähem kordi magatud kui 2001 aastal.

```
SELECT nimi
FROM Ase AS A
WHERE (SELECT Count(*) AS arv FROM Magamine AS M WHERE
A.ase_id=M.ase_id AND Year(algus)=2000)<
(SELECT Count(*) AS arv FROM Magamine AS M WHERE
A.ase_id=M.ase_id AND Year(algus)=2001);
```

WHERE klauslis asuvas võrdluspredikaadis võib nii vasakul kui ka paremal pool võrdlusmärki olla alampäring. Kumbki alampäring tagastab täpselt ühe väärtuse. Kumbki on korreleeruv alampäring, mis leiab vastavalt 2000 ja 2001 aastal alanud magamised.

50. Millisel asemel on magaja, kelle perenimi on Juurikas, kokku kõige kauem maganud?

Esimese päringuga leiame me kui kaua on Juurikas igal asemel maganud.

```
SELECT nimi AS aseme_nimi, Sum(kestus) AS pikkus
FROM Magaja INNER JOIN (Ase INNER JOIN Magamine ON Ase.ase_id =
Magamine.ase_id) ON Magaja.magaja_id = Magamine.magaja_id
WHERE perenimi='Juurikas'
GROUP BY Magaja.magaja_id, Magaja.perenimi, Ase.nimi;
```

Salvestame päringu nime all *Juurika_magamised.*

```
SELECT aseme_nimi
FROM Juurika_magamised
WHERE pikkus=(SELECT Max(Pikkus) AS maks FROM
Juurika_magamised);
```

MS Accessi spetsiifiline lahendus, mida ei soovita kasutada:

```
SELECT TOP 1 nimi AS aseme_nimi
FROM Magaja INNER JOIN (Ase INNER JOIN Magamine ON Ase.ase_id =
Magamine.ase_id) ON Magaja.magaja_id = Magamine.magaja_id
WHERE perenimi='Juurikas'
GROUP BY Magaja.magaja_id, Magaja.perenimi, Ase.nimi
ORDER BY Sum(kestus) DESC;
```

51. Millisel kuul sündinud magajad on kõige suuremad unimütsid? (nende keskmine magamise kestus on kõige suurem?)

```
SELECT Month(synni_aeg) AS kuu, Avg(kestus) AS keskmine
FROM Magaja INNER JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Month(synni_aeg);
```

Salvestan nime all *Statistika_kuude_kaupa*

```
SELECT kuu, keskmine
FROM Statistika_kuude_kaupa
WHERE keskmine=(SELECT Max(keskmine) AS maks FROM
Statistika_kuude_kaupa);
```

52. Teha statistikat selle kohta kui palju on mingi aasta mingis kuus magamisi registreeritud? Vaadelda aastaid alates 2000. Tulemuses peaks olema kuu number, aasta number ja selle aasta vastaval kuul alanud magamiste arv. Kui mingi aasta mingis kuus pole alanud ühtegi magamist, siis sellise kuu kohta ei pea tulemuses rida olema.

```
SELECT Month(algus) AS kuu, Year(algus) AS aasta, Count(*) AS arv
FROM Magamine
WHERE Year(algus)>=2000
GROUP BY Month(algus), Year(algus)
ORDER BY  Year(algus), Month(algus);
```

53. Leia kõik magajad, kellel on täna sünnipäev. Esitage tulemuses magaja nimi kujul kus kõigepealt on eesnime esitäht, seejärel punkt ja tühik ning perekonnanimi.

```
SELECT Left(eesnimi,1) & '. ' & perenimi AS sünnipäevalaps
FROM Magaja
WHERE Month(synni_aeg)=Month(Date()) AND
                        Day(synni_aeg)=Day(Date());
```

54. Leia kõik magajad, kellel on sellel aastal sünnipäev juba ära olnud! Leia ka nende vanus aastates (täisosa).

```
SELECT Magaja.*, DateDiff("yyyy",synni_aeg,date()) AS vanus
FROM Magaja
WHERE Month(synni_aeg)<Month(Date())OR
(Month(synni_aeg)=Month(Date()) AND Day(synni_aeg)<Day(Date()));
```

Sünnikuu peab olema väiksem kui käesolev kuu või sünnikuu ja käesolev kuu on võrdsed, aga sünni päeva näitav arv on väiksem kui käesolevat päeva näitav arv.

55. Leia kõik magajad, kellel ei ole sellel aastal veel sünnipäeva ära olnud! Leia ka nende vanus aastates (täisosa).

```
SELECT Magaja.*, DateDiff("yyyy",synni_aeg,date())-1 AS vanus
FROM Magaja
WHERE  NOT  (Month(synni_aeg)<Month(Date())OR
(Month(synni_aeg)=Month(Date()) AND Day(synni_aeg)<Day(Date())));
```

Andmed magaja vanuse kohta aastates (täisosa) annab päring:

```
SELECT Magaja.*, DateDiff("yyyy",synni_aeg,date()) AS vanus
FROM Magaja
WHERE Month(synni_aeg)<Month(Date())OR
(Month(synni_aeg)=Month(Date()) AND Day(synni_aeg)<Day(Date()))

UNION SELECT Magaja.*, DateDiff("yyyy",synni_aeg,date())-1 AS vanus
FROM Magaja
WHERE  NOT  (Month(synni_aeg)<Month(Date())OR
(Month(synni_aeg)=Month(Date()) AND Day(synni_aeg)<Day(Date())));
```

56. Leia keskmine magamise kestus üle selliste magamiste, kus magaja oli magamise alguses üle 5000 päeva vana.

```
SELECT Avg(kestus) AS keskmine
FROM Magaja INNER JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
WHERE algus>synni_aeg+5000;
```

57. Selleks, et magaja magaks 1 tund kulub Une Matil 150 g liiva. Mitu kg liiva on Une Mati pidanud 2002 aastal ära kulutama. Arvesse lähevad need magamised, mis algavad 2002 aastal.

```
SELECT ((Sum(kestus))/60)*0.15 AS liiva_kogus
FROM Magamine
WHERE Year(algus)=2000;
```

Kestus on antud minutites. Kestus summeeritakse. Tulemus jagatakse 60-ga, et leida kestus tundides. Tundide arv korrutatakse 0.15-ga, et leida kulunud liiva hulk kilogrammides.

58. Mitme protsendi magajate kohta on andmeid rohkem kui 5-e magamise kohta?

```
SELECT (Count(*)*100)/(SELECT Count(*) AS arv FROM Magaja) AS
protsent
FROM
(SELECT magaja_id
FROM Magamine
GROUP BY magaja_id
HAVING Count(*)>5) AS foo;
```

Vaadake ülesande 31 lahenduse kirjeldusest, kuidas vältida nulliga jagamist.

59. Koostage paariviisilised kombinatsioonid asemete nimedest. Üks kombinatsioon peab ilmuma vaid üks kord – st, et "Ase1", "Ase2" on sama kombinatsioon, kui "Ase2", "Ase1".

```
SELECT A1.nimi AS nimi1, A2.nimi AS nimi2
FROM Ase AS A1, Ase AS A2
WHERE A1.nimi<A2.nimi;
```

60. Leia iga magaja kohta tema asetus magajate magamiste kogukestuse pingereas. Tulemuses tuleb esitada 4 veergu: magaja_id, perenimi, magaja magamise kogukestus, asetus pingereas

Lisame antud näites lisatingimuse, et kui magaja magamiste või nende pikkuste kohta andmeid ei ole, siis tuleb arvestada, et pikkus=0.

Näide:

magaja_id	perenimi	kogukestus	asetus
5	Karu	3019	1
4	Juurikas	2962	2
2	Jaan	2923	3
3	Ta?mik	2815	4
6	Vaarikas	2349	5

Teen päringu, mis leiab kõigi magajate magamise kogukestuse. Kui magaja magamiste kohta puuduvad andmed, siis kasutan MS Accessi süsteemi-defineeritud funktsiooni *Nz*, et saada selliste isikute korral tulemuseks kogukestus=0. MS Accessi süsteemi-defineeritud funktsiooni *CInt* läheb vaja, et teisendada *Nz* funktsiooni tulemus täisarvu tüüpi väärtuseks. Muidu osutuvad järgnevate päringute tulemused valeks.

```
SELECT Magaja.magaja_id, perenimi, CInt(Nz(Sum(kestus),0)) AS
kogukestus
FROM Magaja LEFT JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Magaja.magaja_id, perenimi;
```

Salvestan päringu nime all *Kogukestused.*

Lõpptulemuse annab päring:

```
SELECT Ko.magaja_id, Ko.perenimi, Ko.kogukestus,
        (SELECT Count(*) AS arv FROM
        (SELECT DISTINCT kogukestus
        FROM Kogukestused) AS Ko1
        WHERE Ko.kogukestus<=Ko1.kogukestus) AS asetus
FROM Kogukestused AS Ko
ORDER BY Ko.kogukestus DESC;
```

Päringu ideeks on käivitada iga magaja kohta korreleeruv alampäring. See kasutab omakorda alampäringut. Kõige sisemine alampäring leiab ***erinevad*** magamiste kogukestused. Sellest välimine alampäring aga leiab kõigi selliste erinevate kogukestuste arvu, mis on antud magaja magamiste kogukestusest suuremad või sellega võrdsed. See arv on ühtlasi ka isiku asetuseks üldises pingereas.

Oletame, et magamise kogukestused on järgnevad:

1000
1000
100
100

Sellisel juhul on siin nimekirjas 2 järku (see on erinevate kogukestuste arv).

1000	1
1000	1
100	2
100	2

PostgreSQLis ja Oracles saab sellise ülesande lahendada analüütilise funktsiooniga DENSE_RANK(), mille abil saab leida asetuse (järgu) pingereas, kusjuures asetuses ei ole "tühikuid" (asetused on näiteks "1, 2, 2, 3", mitte "1, 2, 2, 4").

```
SELECT Magaja.magaja_id, perenimi, coalesce(Sum(kestus),0) AS
kogukestus,  DENSE_RANK() OVER (ORDER BY
Coalesce(Sum(kestus),0) DESC) AS asetus
FROM Magaja LEFT JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Magaja.magaja_id, perenimi;                    (PO)
```

Kui magaja magamiste kohta puuduvad andmed, siis kasutan süsteemi-defineeritud funktsiooni *coalesce*, et saada selliste isikute korral tulemuseks kogukestus=0.

61. Leia magajad, kes on maganud kõigil asemetel.

Ülesannete lahendamisel on üheks kasulikuks võtteks ülesande ümbersõnastamine. Antud juhul võiks küsida: "Leia kõik magajad, kelle puhul ei leidu aset, millel ta ei oleks maganud." Siin on tegemist kahekordse eitusega.

Lahendus 1.

```
SELECT magaja_id, eesnimi, perenimi
FROM Magaja AS M
WHERE NOT EXISTS
(SELECT *
FROM Ase AS A
WHERE NOT EXISTS (
    SELECT *
    FROM Magamine AS MM
   WHERE MM.magaja_id=M.magaja_id AND MM.ase_id=A.ase_id));
```

Veel üks võimalik ümbersõnastamine: "Leia kõik magajad, kes on maganud nii mitmel erineval asemel kui tabelis Ase on asemeid".

Lahendus 2.

```
SELECT M2.magaja_id, M2.eesnimi, M2.perenimi
FROM  (SELECT DISTINCT magaja_id, ase_id
FROM Magamine) AS M1, Magaja M2
WHERE M1.magaja_id=M2.magaja_id
GROUP BY M2.magaja_id, M2.eesnimi, M2.perenimi
HAVING COUNT(*)=(SELECT COUNT(*) AS arv FROM Ase);
```

Sinisega tähistatud alampäringuga leitakse erinevad asemed, millel magaja on maganud. Iga magaja kohta leitakse, kui mitmel ***erineval*** asemel on ta maganud. Juhul, kui see arv on võrdne kõikide asemete arvuga, siis järelikult on ta maganud kõigil asemetel. Antud lahendus arvestab sellega, et tabelis *Ase* on deklareeritud võti ja seal ei ole korduvaid ridu. Vastasel juhul *ei annaks* alampäring *SELECT COUNT(*) AS arv FROM Ase* tulemuseks kõikide asemete arvu.

Lahendus 3.

```
SELECT M2.magaja_id, M2.eesnimi, M2.perenimi
FROM  Magaja AS M2
WHERE (SELECT Count(*) AS arv FROM  (SELECT DISTINCT magaja_id,
ase_id
FROM Magamine) AS M1 WHERE M1.magaja_id=M2.magaja_id)
=(SELECT COUNT(*) AS arv FROM Ase);
```

Lahendused 2 ja 3 lähtuvad eeldusest, et tabelis *Ase* ei leidu korduvaid ridu. Tuleb meeles pidada, et SQLis ei ole tabelites võtmete kirjeldamine *kohustuslik* ja põhimõtteliselt võib luua ka tabeli, kus on korduvad read (see on muidugi VÄGA HALB praktika). Näiteandmebaasis on tabelis *Ase* võtmed olemas ja päring *SELECT Count(*) AS arv FROM Ase* leiab *erinevate* asemete arvu.

Antud juhul on tabelis *Magamine* veerg *ase_id* kohustuslik ja see lihtsustab päringu koostamist. Kui see veerg oleks mittekohustuslik, siis võiks tabelis olla magamise andmeid, mille puhul ase pole määratud – st andmebaasi

kasutajale pole teada, millisel asemel magaja magas. Oletame, et magaja on maganud kõigil asemetel peale ühe ja lisaks on tal üks magamine, mille puhul pole aset teada. See ase võis (aga ei pruukinud) olla see, millel magamise kohta andmed puuduvad. Sellisel juhul tuleb otsustada, kuidas puuduva asemega rida arvestada. Kõige lihtsam oleks sellised read vaatluse alt välja jätta (tingimus: ase_id IS NOT NULL).

Lahendus 4.

```
SELECT magaja_id, eesnimi, perenimi
FROM Magaja
WHERE magaja_id NOT IN
(SELECT magaja_id
FROM Magaja AS M, Ase AS A
WHERE NOT EXISTS (SELECT 1 FROM Magamine AS MM WHERE
MM.magaja_id=M.magaja_id AND MM.ase_id=A.ase_id));
```

Sinisega esitatud alampäringuga leitakse sellised *magaja_id* ja *ase_id* paarid, mille korral vastava identifikaatoriga magaja on maganud vastava identifikaatoriga asemel. Punasega esitatud alampäringuga leitakse selliste magajate identifikaatorid kes ei ole kõigil asemetel maganud (nende identifikaatorid leiduvad *magaja_id* ja *ase_id* paaride hulgas, mille korral vastava identifikaatoriga magaja ei ole maganud vastava identifikaatoriga asemel). Nüüd jääbki ainult üle leida sellised magajad, kelle identifikaator ei sisaldu alampäringuga leitud magajate identifikaatorite hulgas.

62. Leia magajad, kes pole maganud kõigil asemetel.

Tegemist on eelmise ülesande lihtsa täiendusega.

```
SELECT M2.magaja_id, M2.eesnimi, M2.perenimi
FROM  Magaja M2 LEFT JOIN (SELECT DISTINCT magaja_id, ase_id
FROM Magamine) AS M1 ON M1.magaja_id=M2.magaja_id
GROUP BY M2.magaja_id, M2.eesnimi, M2.perenimi
HAVING Count(*)<(SELECT Count(*) AS arv FROM Ase);
```

LEFT JOIN (välisühendamise) operatsiooni läheb vaja, sest võib leiduda isikuid, kelle kohta pole andmeid ühegi seotud magamise kohta. Kõigi asemete arv peab olema suurem kui erinevate asemete arv, kus isik on maganud.

Teine lahendus oleks olnud kõigepealt kõik sellised isikud, kes on maganud kõigil asemetel ja siis kõik ülejäänud.

Lahendus 2:

```
SELECT magaja_id, eesnimi, perenimi
FROM Magaja
WHERE magaja_id NOT IN
(SELECT M2.magaja_id
FROM  Magaja AS M2
WHERE (SELECT Count(*) AS arv FROM  (SELECT DISTINCT magaja_id,
ase_id
FROM Magamine) AS M1 WHERE M1.magaja_id=M2.magaja_id)
=(SELECT Count(*) AS arv FROM Ase));
```

63. Leidke millistel asemetel pole magajad maganud. Tulemuses peavad olema magaja identifikaatori ja perenime ning aseme identifikaatori ja perenime paarid.

```
SELECT magaja_id, eesnimi, perenimi, ase_id, nimi
FROM Magaja, Ase
WHERE NOT EXISTS (SELECT * FROM Magamine WHERE
Magamine.magaja_id=Magaja.magaja_id AND
Magamine.ase_id=Ase.ase_id)
ORDER BY magaja_id, ase_id;
```

Põhipäringus leitakse magajate ja asemete paarid (korrutis). Iga paari korral kontrollitakse, kas selline paar eksisteerib mõnes tabeli *Magamine* reas. Kui ***ei eksisteeri*** (NOT EXISTS), siis predikaadi hindamise tulemus on *TRUE* ja vastava paari andmed kuuluvad väljastamisele.

```
SELECT magaja_id, eesnimi, perenimi, ase_id, nimi
FROM Magaja, Ase
WHERE Magaja.magaja_id NOT IN (SELECT magaja_id FROM Magamine
WHERE Ase.ase_id=Magamine.ase_id) OR
Ase.ase_id NOT IN (SELECT ase_id FROM Magamine WHERE
Magaja.magaja_id=Magamine.magaja_id)
ORDER BY magaja_id, ase_id;
```

64. **Tabeli *Magaja* veergu *magaja_id* genereeritakse andmebaasisüsteemi poolt sammuga 1 kasvavate arvude jada alates arvust 1. (1, 2, 3, 4, 5, ...). Öeldakse, et kasvavalt sorteeritud arvujada (a_1, a_2, a_3,...a_{k-1}, a_k, ...) ei ole täielik, kui leidub selline kõrvutiasuvate elementide paar a_k ja a_{k-1}, nii et $a_k - a_{k-1} > 1$. Tehke päring, mis juhul, kui veerus *magaja_id* olev arvujada pole täielik, annab teate "Arvujada pole täielik".**

```
SELECT DISTINCT 'Arvujada pole täielik' AS teade
FROM Magaja
HAVING Count(*)<>Max(magaja_id);
```

Kui nt maksimaalne identifikaator on 18 aga tabelis on 17 rida, siis järelikult on arvujadas tühik.

65. Tehke päring, mis leiab iga magaja kohta, kui mitu protsenti moodustab mingil asemel magamiste arv tema magamiste koguarvust. Tulemuses tuleb näidata magaja perenime ja identifikaatorit, aseme nime ja identifikaatorit, magaja asemel magamiste arvu ning asemel magamiste protsenti. Tulemus peab olema sorteeritud magaja identifikaatori järgi.

Tulemuse näide:

magaja_id	magaja_ perenimi	ase_id	aseme_nimi	arv	protsent
1	Mati	2	Ase2	3	75
1	Mati	5	Ase5	1	25
2	Laasik	1	Ase1	1	12,5
2	Laasik	2	Ase2	1	12,5
2	Laasik	3	Ase3	1	12,5
2	Laasik	4	Ase4	1	12,5
2	Laasik	5	Ase5	1	12,5
2	Laasik	6	Ase6	2	25
2	Laasik	7	Ase7	1	12,5

```
SELECT M.magaja_id, M.perenimi AS magaja_perenimi, A.ase_id, A.nimi
AS aseme_nimi, Count(*) AS arv,
Round(((Count(*)*100)/ (SELECT Count(*) AS arv FROM Magamine AS
Ma1 WHERE M.magaja_id=Ma1.magaja_id)),1) AS protsent
FROM Ase AS A INNER JOIN (Magaja AS M  INNER JOIN Magamine AS
Ma ON M.magaja_id = Ma.magaja_id) ON A.ase_id = Ma.ase_id
GROUP BY M.magaja_id, M.perenimi, A.ase_id, A.nimi
ORDER BY M.magaja_id, A.ase_id;
```

Põhipäring leiab magajate ja asemete andmed ning asemel magamiste arvu. Protsent arvutatakse SELECT klauslisse paigutatud korreleeruva alampäringu abil. Alampäringuga:

```
(SELECT Count(*) AS arv FROM Magamine AS Ma1 WHERE
M.magaja_id=Ma1.magaja_id)
```

leitakse iga põhipäringu rea kohta selles oleva magaja magamiste koguarv. Tulemust kasutatakse protsendiarvutuses ning kõige lõpuks tulemus ümardatakse üks koht peale koma:

```
Round(((Count(*)*100)/ (SELECT Count(*) AS arv FROM Magamine AS
Ma1 WHERE M.magaja_id=Ma1.magaja_id)),1) AS protsent
```

Vaadake ülesande 31 lahenduse kirjeldusest, kuidas vältida nulliga jagamist.

66. **Leia tabelis *Magaja* kõige sagedamini esinev eesnimi (eesnimed).**

Tegemist on moodi leidmise ülesandega. Mood on hulgas kõige sagedamini esinev väärtus. Kui hulgas on kaks sellist väärtust, siis nimetatakse seda bimodaalseks jaotuseks, kui kolm sellist väärtust siis trimodaalne jaotus jne.

Lahendus 1:

```
SELECT eesnimi
FROM (SELECT Magaja.eesnimi, Count(*) AS mood
FROM Magaja
GROUP BY Magaja.eesnimi) AS t
WHERE mood=(SELECT Max(arv) FROM (SELECT Count(*) AS arv FROM
Magaja GROUP BY Magaja.eesnimi) AS t2);
```

Leitakse iga eesnime kohta selle esinemiste arv: **(SELECT Magaja.eesnimi, Count(*) AS mood FROM Magaja GROUP BY Magaja.eesnimi) AS t**
WHERE klauslis määratakse, et tulemuses soovitakse näha vaid selliseid eesnimesid, mille puhul on esinemiste arv maksimaalne.

Lahendus 2 (Celcko 2000):

```
SELECT eesnimi
FROM Magaja
GROUP BY eesnimi
HAVING Count(*)>=ALL (SELECT Count(*) AS arv
    FROM Magaja
    GROUP BY eesnimi);
```

Alampäringuga leitakse iga eesnime kohta seda omavate isikute arv. Põhipäringus leitakse samuti iga eesnime kohta seda omavate isikute arv. Tingimusele vastavad eesnimed, mille puhul põhipäringus leitud isikute arv on suurem ***või võrdne kõigist*** alampäringuga leitud arvudest (suurima arvuga on see võrdne, kõigist teistest arvudest on suurem).

MS Accessi-spetsiifiline lahendus.

```
SELECT DISTINCT TOP 1 eesnimi
FROM Magaja
GROUP BY eesnimi
ORDER BY Count(*) DESC;
```

PostgreSQLis ja Oracles saanuks ülesande lahendamiseks kasutada analüütilist funktsiooni DENSE_RANK().

```
SELECT eesnimi
FROM (SELECT  eesnimi, DENSE_RANK()  OVER (ORDER BY Count(*)
DESC) AS asetus
FROM Magaja
WHERE eesnimi IS NOT NULL
GROUP BY eesnimi) foo
WHERE asetus=1;                                          (PO)
```

67. Tehke päring, millega leiate iga magaja kõik andmed ja lisaks tema kõige hilisema pikkuse ja kaalu mõõtmise tulemuse. Salvestage see päring ja kasutage järgmiste ülesannete lahendamisel.

```
SELECT Magaja.*, Mootmine.pikkus, Mootmine.kaal
FROM Magaja LEFT JOIN (SELECT *
FROM Mootmine
WHERE mootmise_aeg=(SELECT Max(mootmise_aeg) AS maks FROM
Mootmine AS Mo1 WHERE Mo1.magaja_id=Mootmine.magaja_id)
) AS Mootmine2 ON Magaja.magaja_id = Mootmine2.magaja_id;
```

Salvestan päringu nime all *Magaja_mootmistega* (Sisuliselt loon vaate e. virtuaalse tabeli).

Välisühendamist tuleb kasutada, et saada tulemusse magajad, keda pole kordagi mõõdetud.

NB! Kui teha järgnev päring, siis annaks see tulemuseks vaid isikud, keda on vähemalt üks kord mõõdetud:

```
SELECT Magaja.*, Mootmine.pikkus, Mootmine.kaal
FROM Magaja LEFT JOIN Mootmine ON Magaja.magaja_id =
Mootmine.magaja_id
WHERE mootmise_aeg=(SELECT Max(mootmise_aeg) AS maks FROM
Mootmine AS Mo1 WHERE Mo1.magaja_id=Mootmine.magaja_id);
```

Miks selline tulemus? Kõigepealt toimub kahe tabeli ühendamine. Tulemuses on ka read selliste magajate kohta, keda pole mõõdetud. Seejärel rakendatakse WHERE klauslit. Lõpptulemusest jäävad välja read, kus ühendamise tulemuses mõõtmistulemuste väljades väärtused puudusid.

68. Sobiva kehakaalu arvutamiseks on mitmeid valemeid. Üks nendest on "Kehamassi indeks" (http://www.halls.md/body-mass-index/bmirefs.htm)

See leitakse: Kehamassi indeks= (kehakaal kilogrammides)/(keha pikkus meetrites)^2

Täiskasvanu	Naine	Mees
anoreksia	< 17.5	
alakaaluline	<19.1	<20.7
normaalse kaaluga	19.1-25.8	20.7-26.4
veidi ülekaaluline	25.8-27.3	26.4-27.8
ülekaaluline	27.3-32.3	27.8-31.1
tõsiselt ülekaaluline	>32.3	>31.1
haiglaslikult ülekaaluline	35 – 40	
superülekaaluline	40 – 50	
...	50 – 60	

Tehke päring, mis leiab kõigi magajate identifikaatori, eesnime, perenime, soo ja kehamassi indeksi. Lisage päringusse piirang, et näidataks ainult normaalkaalust suurema kaaluga inimesi. Pange tähele, et meestel ja naistel on erinevad indeksi vahemikud.

```
SELECT Magaja.magaja_id, eesnimi, perenimi, sugu, kaal/(pikkus/100)^2
AS indeks
FROM Magaja_mootmistega
WHERE (kaal/(pikkus/100)^2>25.8 AND sugu='N') OR (kaal/
(pikkus/100)^2>26.4 AND sugu='M');
```

Andmebaasisüsteemis, mis võimaldab kasutada funktsioonil baseeruvaid indekseid tuleks selle päringu kiirendamiseks luua indeks avaldise "kaal/(pikkus/100)^2" põhjal.

69. **Tabelis *HinnangA* on inimese pikkuse vahemiku algus ja lõpp-punkt ning hinnang sellesse vahemikku jäävale pikkusele. Tabelis *HinnangB* on sama informatsioon, kuid see on esitatud teisel kujul. Iga pikkuse hinnangu kohta on antud suurim pikkus, millega seda hinnangut võib saada. Ülesandeks on koostada SQL lause, mis leiab pikkuse hinnangu igale isikule, kellel on pikkus määratud. Tuleb esitada kaks lahendusvarianti, kus ühel juhul on hinnangu leidmiseks kasutatud tabelit *HinnangA* ja teisel juhul tabelit *HinnangB*. Kummalgi juhul peaks tulemuses olema neli veergu: *magaja_id*, *perenimi*, *pikkus*, *hinnang*.**

Päring, mis kasutab tabelit *HinnangA*.

```
SELECT magaja_id, perenimi, pikkus, hinnang
FROM HinnangA AS H, Magaja_mootmistega AS M
WHERE M.pikkus BETWEEN H.madalaim AND H.korgeim;
```

Ühes päringus on koos nii equijoin kui ka non-equijoin.

Päringud, mis kasutavad tabelit *HinnangB*. Leitakse magaja pikkusest väiksemate hinnangu vahemike alguste seast maksimaalne.

```
SELECT magaja_id, perenimi,  pikkus, Max(HinnangB.madalaim) AS
hinnang
FROM Magaja_mootmistega, HinnangB
WHERE (HinnangB.madalaim<=pikkus)
GROUP BY magaja_id, perenimi, pikkus;
```

Salvestan nime all *Pikkuse_hinnang*. Lõpptulemuse annab päring:

```
SELECT magaja_id, perenimi, pikkus, HinnangB.hinnang
FROM Pikkuse_hinnang INNER JOIN HinnangB ON
Pikkuse_hinnang.hinnang = HinnangB.madalaim;
```

Päringud võib panna kokku ka üheks:

```
SELECT magaja_id, perenimi, pikkus, HinnangB.hinnang
FROM (SELECT magaja_id, perenimi, pikkus, Max(HinnangB.madalaim)
AS hinnang
FROM Magaja_mootmistega, HinnangB
WHERE (HinnangB.madalaim<=pikkus)
GROUP BY magaja_id, perenimi, pikkus) Pikkuse_hinnang INNER JOIN
HinnangB ON Pikkuse_hinnang.hinnang = HinnangB.madalaim;
```

70. Leia magajad, kelle pikkus on suurem kui pikima aseme pikkus.

Lahendus 1:

```
SELECT *
FROM Magaja_mootmistega
WHERE pikkus>ALL(SELECT pikkus FROM Ase WHERE pikkus IS NOT
NULL);
```

Lahendus 2:

```
SELECT *
FROM Magaja_mootmistega
WHERE pikkus>(SELECT Nz(Max(pikkus),0) AS maks FROM Ase);
```

Lahendus 1 ja 2 ei ole päris samaväärsed. Erinevus tekib siis, tabelis *Ase* ei ole ühtegi rida. Lahendus 1 annab sellisel juhul tulemuseks kõigi magajate andmed – isegi selliste magajate andmed, kelle kohta ei ole teada ühegi pikkuse mõõtmise tulemus. Samas lahendus 2 annab sellisel juhul tulemuseks ainult selliste magajate andmed, kellel on pikkus määratud.

Milleks läheb lahenduses 2 vaja funktsiooni *Nz*? See funktsioon tagab, et kui tabelis *Ase* ei ole ühtegi rida, siis on alampäringu tulemus 0, mitte NULL. *Nz* funktsioon võimaldab asendada NULLi mingi väärtusega. Kui me ei kasutaks funktsiooni *Nz* ja tabelis *Ase* ei oleks ühtegi rida, siis ei oleks lahendus 2 päringu tulemuses mitte ühtegi rida.

71. Leia kõik asemed, mis on pikemad kui asemed, mis on vähemalt 150 cm laiad.

```
SELECT *
FROM Ase
WHERE pikkus>(SELECT Max(pikkus) AS maks
FROM Ase
WHERE laius>=150);
```

72. Leia magajad, kes on kõige lühemal asemel maganud magajatest viimase mõõtmise tulemusena kõige pikemad.

Järgnev päring leiab magajad, kes on maganud kõige lühemal asemel.

```
SELECT *
FROM Magaja_mootmistega
WHERE magaja_id IN
(SELECT magaja_id
FROM Ase INNER JOIN Magamine ON Ase.ase_id=Magamine.ase_id
WHERE pikkus=(SELECT Min(pikkus) FROM Ase));
```

Salvestan nime all *Lühimal_asemel_maganud*. Järgnev päring leiab kõige lühemal asemel maganud magajate seast kõige pikema.

```
SELECT *
FROM Lühimal_asemel_maganud
WHERE pikkus=(SELECT Max(pikkus) AS maks FROM
Lühimal_asemel_maganud);
```

73. Leia iga magaja kohta temast kuni 10% kergemad ja raskemad isikud. Isiku võrdlust ise-endaga ei tule näidata. Esitada tulemuses isikute paarid, kus on näidatud kummagi isiku kohta: magaja_id, perenimi ja kaal. Tulemuse näide:

magaja_id	perenimi	kaal	võrreldava_id	võrreldava_ perenimi	võrreldava_kaal
5	Juurikas	125	1	Mati	130
5	Juurikas	125	7	Vain	120
6	Karu	158	2	Laasik	157
2	Laasik	157	6	Karu	158
1	Mati	130	5	Juurikas	125
1	Mati	130	7	Vain	120
7	Vain	120	5	Juurikas	125
7	Vain	120	1	Mati	130

Võib kasutada tabeli ühendamist iseendaga (*self join*). Tulemus on parema loetavuse huvides sorteeritud.

```
SELECT M.magaja_id, M.perenimi, M.kaal, M1.magaja_id AS võrreldava_id,
M1.perenimi AS võrreldava_perenimi, M1.kaal AS võrreldava_kaal
FROM Magaja_mootmistega AS M, Magaja_mootmistega AS M1
WHERE (M.magaja_id<>M1.magaja_id) AND (M1.kaal BETWEEN
M.kaal*0.9 And M.kaal*1.1)
ORDER BY M.perenimi, M1.perenimi;
```

74. Leidke, kui mitu sentimeetrit on magaja pikemaks või lühemaks jäänud kui võrrelda tema kõige viimast pikkuse mõõtmist kõige esimese pikkuse mõõtmisega. Päringu tulemuses peaks olema magaja identifikaator, perenimi ja pikkuse muudatus. Päringu

tulemuses peavad olema andmed ainult selliste magajate kohta, kelle pikkus on muutunud.

```
SELECT magaja.magaja_id, perenimi, hiliseim.pikkus-varaseim.pikkus AS
Muutus
FROM ((SELECT magaja_id, pikkus
FROM Mootmine Mo
WHERE Mo.mootmise_aeg=(SELECT Min(mootmise_aeg) AS minn FROM
Mootmine M WHERE M.magaja_id=Mo.magaja_id)) AS varaseim INNER
JOIN
(SELECT magaja_id, pikkus
FROM Mootmine Mo
WHERE Mo.mootmise_aeg=(SELECT Max(mootmise_aeg) AS maks FROM
Mootmine M WHERE M.magaja_id=Mo.magaja_id)) AS hiliseim ON
varaseim.magaja_id=hiliseim.magaja_id) INNER JOIN Magaja ON
hiliseim.magaja_id=Magaja.magaja_id
WHERE hiliseim.pikkus-varaseim.pikkus<>0;
```

Punasega tähistatud alampäring leiab magaja kõige varasema pikkuse mõõtmise tulemuse ja sinisega tähistatud alampäring kõige hilisema pikkuse muudatuse.

75. Leidke magajad, kellel on vähemalt 2 seotud magamist. Lisatingimus on, et päringus ei tohi kasutada Count funktsiooni. Päringu tulemuses peavad olema magaja identifikaator, eesnimi ja perenimi.

```
SELECT magaja_id, eesnimi, perenimi
FROM Magaja
WHERE magaja_id IN
(SELECT M1.magaja_id
FROM Magamine AS M1, Magamine AS M2
WHERE M1.magaja_id=M2.magaja_id AND M1.algus<>M2.algus);
```

Alampäring kasutab *self joini* (tabeli ühendamist iseendaga) ning leiab magajad, kellel on vähemalt kaks erinevatel aegadel toimunud magamist. Põhipäringuga leitakse selliste magajate andmed tabelist *Magamine.* Päringu lahendus lähtub võtmete poolt jõustatud kitsendusest, et ühel magajal ei saa ühel ajahetkel alata rohkem kui üks magamine.

76. Sorteerige magamised alguse aja ja aseme numbrilise identifikaatori järgi kasvavalt ning leidke nende kestuse jooksev summa. Päringu tulemuses peab olema aseme (numbriline) identifikaator, nimi, magaja identifikaator, perenimi, magamise kestus, algus ja jooksev kestuste summa. Tulemuse näide:

ase_id	nimi	magaja_id	perenimi	kestus	algus	jooksev_kestus
6	Ase6	3	Ta?mik	325	20.11.2000 21:00:00	325
5	Ase5	1	Vaarikas	411	25.11.2000 21:00:00	736
6	Ase6	2	Jaan	411	27.11.2000 21:00:00	1147
3	Ase3	5	Karu	154	16.12.2000 21:00:00	1301
4	Ase4	5	Karu	504	26.12.2000 21:00:00	1805

Kuidas leida kestuste jooksev summa? Näiteks kollasega tähistatud reas jooksev_kestus= 325+411+411.

```
SELECT Ase.ase_id, Ase.nimi, Magaja.magaja_id, Magaja.perenimi,
Magamine.kestus, Magamine.algus, (SELECT Sum(kestus) AS summa
FROM Magamine AS MM WHERE MM.algus<=Magamine.algus AND
MM.ase_id<=Magamine.ase_id) AS jooksev_kestus FROM Ase INNER
JOIN (Magaja INNER JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id) ON Ase.ase_id = Magamine.ase_id
ORDER BY Magamine.algus, Magamine.ase_id;
```

Jooksva kestuse leidmiseks kasutatakse SELECT klauslis olevat korreleeruvat alampäringut. See summeerib iga magamise kohta selliste magamiste kestused mis on on alanud varem või samal ajal kui see magamine. Kuna magamist identifitseerib *ase_id* ja *algus* kombinatsioon, peab alampäringus olema ka tingimus *MM.ase_id<=Magamine.ase_id*.

77. Leia iga aasta kohta millal, on alanud vähemalt üks magamine, kui mitmel protsendil selle aasta päevadest on alanud vähemalt üks magamine. Ümardage protsent ühe kohani peale koma. Tulemuse näide:

aasta	protsent
2000	1,1
2001	5,8
2002	6
2003	0,5
2004	0,3
2010	0,8

Vihje: Kasutage funktsioone *Format*, *Year*, *DateAdd*, *DateSerial*, *Round*.

```
SELECT Year(alg) AS aasta, Round(Count(*)*100/
(DateAdd('YYYY',1,DateSerial(Year(alg),1,1))-DateSerial(Year(alg),1,1)),1)
AS protsent
FROM (SELECT DISTINCT Format(algus,'YYYY-MM-DD') AS alg FROM
Magamine) AS ap
GROUP BY Year(alg);
```

Alampäringuga tagatakse, et kui ühel päeval on alanud mitu magamist, siis arvestatakse seda päeva vaid üks kord.

Vaatleme lähemalt SELECT klauslis olevat avaldist. Ülesande lahendamiseks kasutatakse süsteemi-defineeritud kuupäevatöötluse funktsioone. Nagu näete võib ühe funktsiooni tulemus olla teise funktsiooni argumendiks.

Avaldis	Selgitus
Year(alg)	Magamise alguse aasta (näiteks 2008)
DateSerial(Year(alg),1,1))	Magamise alguse aasta esimene päev (näiteks 01.01.2008)
DateAdd('YYYY',1,DateSerial(Year (alg),1,1))	Magamise alguse aastale järgneva aasta esimene päev (näiteks 01.01.2009)
(DateAdd('YYYY',1,DateSerial (Year(alg),1,1)) -DateSerial(Year(alg),1,1))	Magamise alguse aasta päevade arv (näiteks 366)
Round(Count(*)*100/(DateAdd ('YYYY',1,DateSerial(Year(alg),1, 1))-DateSerial(Year(alg),1,1)),1)	Päevade arvu protsent ümardatud ühe kohani peale koma.

Vaadake ülesande 31 lahenduse kirjeldusest, kuidas vältida nulliga jagamist.

78. **Tehke päring, millega leiate tabelist *Magaja* neli juhuslikult valitud rida.**

Vihje: http://www.techrepublic.com/blog/how-do-i/how-do-i-retrieve-a-random-set-of-records-in-microsoft-access/

Lahenduse põhimõte – igale leitud reale lisatakse juhuslik arv. Read sorteeritakse selle arvu järgi ja leitakse esimesed neli.

Lahendus MS Accessis.

```
SELECT TOP 4 *
FROM (SELECT *
FROM Magaja
ORDER BY Rnd(magaja_id)) AS foo;                                    (A)
```

Rnd on juhusliku leidmiseks mõeldud funktsioon.

Lahendus PostgreSQLis.

```
SELECT *
FROM (SELECT *
FROM Magaja
ORDER BY random()) AS foo
LIMIT 4;                                                    (P)
```

Lahendus Oracles.

```
SELECT *
FROM (SELECT *
FROM Magaja
ORDER BY dbms_random.value)
WHERE ROWNUM <= 4;                                          (O)
```

ROWNUM pseudoveerg (väärtus näitab rea tabelist lugemise järjekorranumbrit) ja LIMIT klausel on vastavalt Oracle ja PostgreSQL SQL mägimurraku keelekonstruktsioonid. SQL standard neid ette ei näe.

Järgnev süntaks (FETCH FIRST n ROWS ONLY) on kirjeldatud SQL standardis (SQL:2011). ONLY – väljastatavate ridade arv ei ületa küsitut.

```
SELECT *
FROM (SELECT *
FROM Magaja
ORDER BY random()) AS foo
FETCH FIRST 4 ROWS ONLY;                                    (P)
```

```
SELECT *
FROM (SELECT *
FROM Magaja
ORDER BY dbms_random.value)
FETCH FIRST 4 ROWS ONLY;                                    (O)
```

79. Tehke päring, mille tulemuseks on histogramm, mis näitab kui palju on mingil magajal seotud magamisi. Päringu tulemuses peavad olema andmed vaid selliste magajate kohta (magaja_id, perenimi), kellel on vähemalt üks seotud magamine. Tulemuse näide:

magaja_id	perenimi	histogramm
1	Vaarikas	******
2	Jaan	********
3	Ta?mik	********
4	Juurikas	*******
5	Karu	*******
6	Vaarikas	******
7	Jaaniste	***
8	Mänd1	****
9	Jansen	**
16	Roi	*

Tärnide arv veerule *histogramm* vastavas väljas näitab magajaga seotud magamiste arvu.

Vihje: Kasutage stringifunktsioone *space* (tagastab stringi, milles on etteantud arv tühikuid) ning *replace* (võimaldab stringis märke asendada).

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

Järgnev päring leiab iga magaja (kellel on vähemalt üks seotud magamine) magamiste arvu.

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

Nimetatud arv on argumendiks funktsiooni *space* poole pöördumisel (*space(Count(*))*). Funktsioon tagastab tühikutest koosneva stringi milles olevate tühikute arv võrdub funktsiooni argumendiks oleva arvuga. Tühikutest koosnev string on omakorda üheks argumendiks funktsiooni *replace* poole pöördumisel (replace(space(Count(*)),' ','*')). Funktsioon tagastab stringi kus on asendatud stringi kuuluvad märgid. Esimene argument on string kus tehakse asendusi, teine argument on asendatav märk (antud juhul tühik) ning kolmas argument on märk mida soovitakse asendusena kasutada (antud juhul "*").

80. **Leia magajad, kes on kõigi magajate seas vanuselt kolmandal kohal. Lahenda ülesanne kahel viisil – kasutades *Count* funktsiooni ja mitte kasutades *Count* funktsiooni. Arvestage päringus vaid selliseid magajaid, kelle sünni aeg on teada.**

Lahendus *Count* funktsiooni kasutamisega.

```
SELECT *
FROM Magaja AS M
WHERE 2=(SELECT Count(*) FROM Magaja AS M2 WHERE
M2.synni_aeg<M.synni_aeg)
AND synni_aeg IS NOT NULL;
```

```
SELECT M5.*
FROM Magaja AS M5, Magaja AS M6, Magaja AS M7
WHERE M5.synni_aeg>M6.synni_aeg AND M6.synni_aeg>M7.synni_aeg
AND M5.magaja_id NOT IN
(SELECT M1.magaja_id
FROM Magaja AS M1, Magaja AS M2, Magaja AS M3, Magaja AS M4
WHERE M1.synni_aeg>M2.synni_aeg AND M2.synni_aeg>M3.synni_aeg
AND M3.synni_aeg>M4.synni_aeg);
```

Alampäringuga leitakse magajad, kellest on vähemalt 3 vanemat magajat. Põhipäringus leitakse magajad, kellest on vähemalt kaks vanemat magajat ja kelle identifikaator ei sisaldu alampäringuga leitud magajate hulgas. Selline magaja saabki olla ainult vanuselt kolmas.

Töökiiruse seisukohalt on see lahendus oluliselt halvem, sest hõlmab otsekorrutiste loomist.

81. Koosta aruande päring kõigi magajate magamise arvude kohta. Esimeses veerus on sulgudes magaja identifikaator ja seejärel magaja perenimi. Teises veerus on selle magaja magamiste arv. Päringu viimases reas on esitatud kõigi magamiste koguarv. Tulemus näide:

magaja	magamiste_arv
(12)Smith	0
(15)Ott	1
(20)Johanson	0
(29)Johanson	2
(30)Ott	0
KOKKU	3

Lahendus.

```
SELECT '(' & Magaja.magaja_id & ')' & perenimi AS magaja,
Count(Magamine.magaja_id) AS magamiste_arv
FROM Magaja LEFT JOIN Magamine ON MAGAJA.magaja_id =
Magamine.magaja_id
GROUP BY '(' & Magaja.magaja_id & ')' & perenimi

UNION SELECT 'KOKKU', Count(magaja_id) AS arv
FROM Magamine;
```

Esimese päringuga leitakse iga magaja magamiste arv. Tulemusele lisatakse UNION operatsiooni abil andmed magamiste koguarvu kohta.

82. Koostage SQL lause, mis leiab kõikvõimalikud asemete ja magajate paarid ning näitab iga paari kohta, kui mitu korda on magaja sellel asemel maganud. Kui ta pole seda kordagi teinud, siis on selles veerus väärtus "0". Tulemuses peab olema 3 veergu – magaja identifikaator, aseme identifikaator ja magaja magamiste arv sellel asemel. Sorteerida tulemus magaja ja aseme identifikaatorite järgi. Tulemuse näide:

magaja	ase	magamiste_arv
12	1	1
12	2	1
12	3	1
12	4	0
12	5	0
13	1	0
13	2	1
13	3	0
13	4	1
13	5	0

Lahendus.

```
SELECT M.magaja_id AS magaja, A.ase_id AS ase, Count(*) AS
magamiste_arv
FROM Magaja AS M INNER JOIN (Ase AS A INNER JOIN Magamine AS
Ma ON A.ase_id = Ma.ase_id) ON M.magaja_id = Ma.magaja_id
GROUP BY M.magaja_id, A.ase_id

UNION SELECT M1.magaja_id, A1.ase_id, 0
FROM Magaja AS M1, Ase AS A1
WHERE NOT EXISTS
(SELECT * FROM Magamine AS Ma1 WHERE
M1.magaja_id=Ma1.magaja_id AND A1.ase_id=Ma1.ase_id)
ORDER BY magaja, ase;
```

Päringu esimese poolega leitakse selliste magajate ja asemete paarid, kus magaja on antud asemel maganud. Samuti leitakse asemel magamiste arv.

Tulemus ühendatakse (UNION) kokku sellise päringu tulemusega, mis leiab magaja ja aseme paarid, kus magaja ei ole asemel maganud. Selliste paaride puhul peab veerus *magamiste_arv* olema väärtus "0". See väärtus on teise SELECT lausesse konstandina sisse kirjutatud.

Lahendus 2:

```
SELECT M.magaja_id AS magaja, A.ase_id AS ase, (SELECT Count(*)
FROM Magamine AS Ma WHERE Ma.magaja_id=M.magaja_id AND
Ma.ase_id=A.ase_id) AS magamiste_arv
FROM Magaja AS M, Ase AS A
ORDER BY magaja_id, ase_id;
```

83. Teha päring, mis leiab isikute arvud magamiskordade kaupa. Päringu tulemuses on kaks veergu. Esimeses veerus on magamiskordade arv (1, 2, 3, 4, 5 jne.) ning teises veerus on isikute arv, kes on nii mitu korda maganud. Magamiskordade arvus on lubatud augud – st kui keegi pole maganud 7 korda, siis ei pea sellist rida tulemuses olema. Kui leidub isikuid, kelle kohta pole andmeid ühegi magamise kohta,

siis tuleb tulemuses neid arvestada magamiskordade arvuga "0". Tulemuse näide:

magamiskordade_arv	isikute_arv_kes_on_nii_mitu_korda_maganud
0	4
1	4
2	5
8	7
9	2
11	1

Lahendus.

```
SELECT magamiskordade_arv, Count(magaja_id) AS
isikute_arv_kes_on_nii_mitu_korda_maganud
FROM
(SELECT Magaja.magaja_id, Count(Magamine.magaja_id) AS
magamiskordade_arv
FROM Magaja LEFT JOIN Magamine ON Magaja.magaja_id =
Magamine.magaja_id
GROUP BY Magaja.magaja_id) AS foo
GROUP BY magamiskordade_arv;
```

Alampäringuga leitakse iga isiku magamiskordade arv. Välisühendamise kasutamine tagab, et leitakse ka isikud, kelle magamiskordade arv=0. Põhipäringus grupeeritakse alampäringu tulemust magamiskordade arvu järgi ja loetakse kokku iga leitud grupi liikmete arv.

84. Leia magajate paarid, kes 2001 aasta jooksul ei alustanud magamist ühesugustel asemetel. Seega, kui nt isik A magas asemetel 1 ja 2 ning isik B asemetel 3 ja 4 ja isik C magas asemetel 2 ja 3 tuleb moodustada paar A ja B kuid mitte paarid A-C ning B-C. Korduvaid paare ei tule näidata. Seega, kui on 1 ja 2, siis 2 ja 1 ei pea tulemuses olema.

Lahendus.

```
SELECT M3.magaja_id, M4.magaja_id AS magaja2_id
FROM Magaja AS M3, Magaja AS M4
WHERE
M3.magaja_id<M4.magaja_id AND  NOT EXISTS
(SELECT M.magaja_id, M2.magaja_id
FROM  (SELECT magaja_id, ase_id FROM Magamine WHERE
Year(algus)=2001) AS M,
(SELECT magaja_id, ase_id FROM Magamine WHERE Year(algus)=2001)
AS M2
WHERE M.magaja_id<M2.magaja_id
AND M.ase_id=M2.ase_id AND M3.magaja_id=M.magaja_id AND
M4.magaja_id=M2.magaja_id)
ORDER BY M3.magaja_id, M4.magaja_id;
```

Alampäringuga leitakse paarid, kus mõlemad osalised on 2001 aastal samal asemel magamist alustanud. Põhipäringuga leitakse sellised paarid, mis alampäringuga leitud paaride hulka ei kuulu.

85. Leidke aseme pikkuse vahemikud (10 cm suurusega) ning iga vahemiku kohta, kui mitu aset sellesse vahemikku kuulub. Koostage vahemikud pikkuste kohta 150 kuni 300 cm (otspunktid kaasa arvatud). Pikkused alla 150 cm peavad moodustama eraldi vahemiku. Pikkused üle 300 cm peavad moodustama eraldi vahemiku. Tulemuse näide:

pikkuse_vahemik	arv
	2
190:199	1
200:209	1
220:229	1
230:239	3
250:259	1

Näiteks antud tulemuse teise rea kohaselt on andmebaasis üks ase, mille pikkus jääb vahemikku 190 ja 199 cm (otspunktid kaasa arvatud).

Vihje: Kasutage funktsiooni *Partition*.
http://office.microsoft.com/en-us/access-help/partition-function-HA001228892.aspx

Lahendus.

```
SELECT Partition(pikkus, 150,300,10) AS pikkuse_vahemik, Count(*) AS arv
FROM Ase
GROUP BY Partition(pikkus, 150,300,10);
```

Partition(pikkus, 150,300,10) – vahemikeks jagamine toimub veerus *pikkus* olevate väärtuste alusel. Vahemikud koostatakse pikkuste kohta 150 kuni 300 ja vahemiku suurus on 10 cm. Pikkused alla 150 cm arvestatakse tulemuses eraldi vahemikuks. Pikkused üle 300 cm arvestatakse tulemuses eraldi vahemikuks.

8. Kasutatud materjalid

1. Celko, J., 2000. *SQL for smarties: advanced SQL programming*. 2nd ed. Academic Press. 553 p.
2. Celko, J., 2005. *Joe Celko's SQL Programming Style*. Morgan Kaufmann Publishers. 216 p.
3. Date, C.J., 2006. *Date on Database. Writings 2000-2006.* Apress. 539 p.
4. Date, C.J., 2009. *SQL and Relational Theory. How to Write Accurate SQL Code*. O'Reilly. 404 p.
5. Date, C.J. & Darwen, H., 1997. *Guide to the SQL Standard.* 4th Revised edition. Pearson Education Limited. 522 p.
6. Chappell, D. & Trimple, J.H. Jr., 2001. *A Visual Introduction to SQL*. Second Edition. Wiley Computer Publishing. 291 p.
7. Gulutzan, P. & Pelzer, T., 1999. *SQL-99 Complete, Really*. Miller Freeman. 1078 p.
8. SQL Course. http://www.sqlcourse2.com (22.02.2001)
9. Michalevicz, Z, & Fogel, D, B., 2000. *How to Solve It: Modern Heuristics*. Springer. 467 p.
10. Molinaro, A., 2005. *SQL Cookbook*. O'Reilly.
11. http://www.cs.mcgill.ca/~kemme/cs421/lectures/421-sql1.pdf (09.07.2005)