

Ülesanne 7

Lahendage palun igaüks seda ülesannet individuaalselt – nii jõuate lühema ajaga rohkem tehtud. Kui teete projekti mitmekesi, siis on vaja omavahel kooskõlastada, kes millised vaated valmis teeb.

Hindamismudeli kohaselt tuleb andmebaasis luua virtuaalne andmete kiht, mille abil andmebaasi kapseldada. Lugege selle kohta slaidide kataloogis olevast failist **Andmebaasi_kapseldamine_IDU0230_2018.ppt**. Vaated on osa sellest kihist. Hindamismudel ootab, et Teie loodav andmebaasirakendus kasutaks andmebaasi läbi virtuaalse andmete kihi (st kui rakendus teeb päringuid otse baastabelite põhjal, siis kaotate punkte). Hindamismudeli lisapunktide osas pakutakse võimalust saada teatud vaadete eest lisapunkte.

Hindamismudel toob eraldi välja, et projektis tuleb luua **vähemalt kolm vaadet (sõltumata projekti tegijate arvust)**. Esimeses järjekorras realiseerige andmete lugemise andmebaasioperatsioonid, millele viidatakse teie poolt projektis realiseeritavat töökohta kirjeldavates kasutusjuhtudes. Kui selleks läheb vaja vähem kui kolm vaadet, siis looge vaateid ka teiste töökohtade jaoks. Teiste töökohtade jaoks vajalike vaadete realiseerimine ei tähenda, et peaksite realiseerima rakenduses need teised töökohad. Rakenduses tuleb ikkagi realiseerida üks töökoht.

Looge esmajärjekorras vaated, mida läheb vaja valitud töökoha realiseerimisel. Kui selleks läheb vaja vähem kui kolm vaadet, siis looge vaateid ka teiste töökohtade jaoks.

Vaade on tabel (virtuaalne tabel). Selles dokumendis minnakse kaasa SQLi tavapärase terminikasutusega ning räägitakse *vaadetest* (virtuaalsed tabelid) ja *tabelitest* (baastabelid).

Ülesande lahenduseks peate tegema kõike järgnevat:

- ⌘ looma CREATE VIEW lauseid kasutades vaated (kuna kõik projekti osalised saavad seda ülesannet lahendada samaaegselt, siis ei tohi loodavad vaated langeda kokku teiste projekti osaliste loodud vaadetega),
- ⌘ kirjutama nende vaadete vabatekstilised kommentaarid ning registreerima need andmebaasi süsteemikataloogi (COMMENT lausetega). Kommentaar peab sõnaliselt kirjeldama, milliseid andmeid see vaade näitab ning milleks läheb seda vaadet vaja,
- ⌘ lisama andmebaasi sellised testandmed, et iga vaate põhjal päringu tegemisel on päringu tulemuses vähemalt üks rida (kui seda pole juba varem tehtud),
- ⌘ kirjutama vaadete katsetamiseks iga loodud vaate kohta SELECT lause, mis leiab vaate kõik andmed,
- ⌘ kirjutama laused loodud vaadete kustutamiseks (need on tuleb samuti projekti dokumendis esitada).

PostgreSQL – CREATE VIEW

<https://www.postgresql.org/docs/11/static/sql-createview.html>

Oracle – CREATE VIEW

https://docs.oracle.com/database/121/SQLRF/statements_8004.htm

PostgreSQL – DROP VIEW

<https://www.postgresql.org/docs/11/static/sql-dropview.html>

Oracle – DROP VIEW

https://docs.oracle.com/database/121/SQLRF/statements_9009.htm

PostgreSQL – COMMENT

<https://www.postgresql.org/docs/11/static/sql-comment.html>

Oracle – COMMENT

https://docs.oracle.com/database/121/SQLRF/statements_4010.htm

Vaate kommenteerimise lause algab COMMENT ON TABLE ...

Kuidas leida päringud, mis tuleks andmebaasis vaadetena realiseerida?

- ✧ Detailanalüüsi kasutusjuhtude tekstikirjeldused. Iga **lugemisoperatsioon** tähendab andmete lugemist ning see lugemine võiks andmebaasis toimuda vaate (või ka tabelifunktsiooni või hetktõmmise) põhjal. Esimeses järjekorras vaadake kasutusjuhte, mis kirjeldavad projektis realiseeritava töökoha funktsionaalsust.

Eeltingimused: Õppejõud on identifitseeritud. Vastuvõtt on registreeritud.

Järeltingimused: Vastuvõtuga seotud vastuvõtuaeg on registreeritud.






Stsenarium (tüüpiline sündmuste järjestus):

- Õppejõud soovib lisada uue vastuvõtuaja.
- **Süsteem** kuvab nimekirja õppejõuga seotud aktiivsetest vastuvõttudest – need on vastuvõttud, mis toimuvad vastuvõtu lisamise ajaga samal kuupäeval või tulevikus. Nimekirjas esitatakse vastuvõtu kuupäev ning vastuvõtu toimumise koht – hoone ja ruum. Vastuvõttud on sorteeritud kuupäeva järgi kahanevalt (**OP2.1**).

OP2.1 põhjal võiks luua vaate. Kui seda operatsiooni kasutatakse mitmes kasutusjuhus, siis toimub pöördumine selle vaate poole kõigile nendele kasutusjuhtudele vastavates rakenduse osades.

- ✧ Vaadake enda MS Accessis tehtud prototüüpi. Iga seal defineeritud SELECT lause põhjal tuleks defineerida vaade.

Queries

-  Aktiivsed_vastuvotud
-  Koik_vastuvotud
-  Mitteaktiivsed_vastuvotud
-  Ruumid
-  Vastuvotuajad_koos_tulijatega

Nende *Queride* põhjal tuleks luua vaated.

Uurige oma prototüübi vorme ja aruandeid. Võib juhtuda, et päring on defineeritud vormi või vormi elemendi (nt liitboks) omadusena. Selle põhjal tuleks luua vaade.

Oppejou_aktiivsed_vastuvotud

Form Header

Lisa uus vastuvõtt Sulge

kuupaev

Detail

Vaata/muuda kuupaev

Property Sheet

Selection type: Form

Form

Format Data Event Other All

Record Source	SELECT Vastuvott.kuupaev, Ruum.hoone_kood, Ruum.ruum
Caption	Aktiivsed vastuvotud
Pop Up	No
Modal	No
Default View	Continuous Forms

Vastuvotu_haldamine

Form Header

Tühista Sulge

Detail

+

vastuvott_id: vastuvott_id

ruum: * ruum_id

Property Sheet

Selection type: Combo Box

ruum_id

Format Data Event Other All

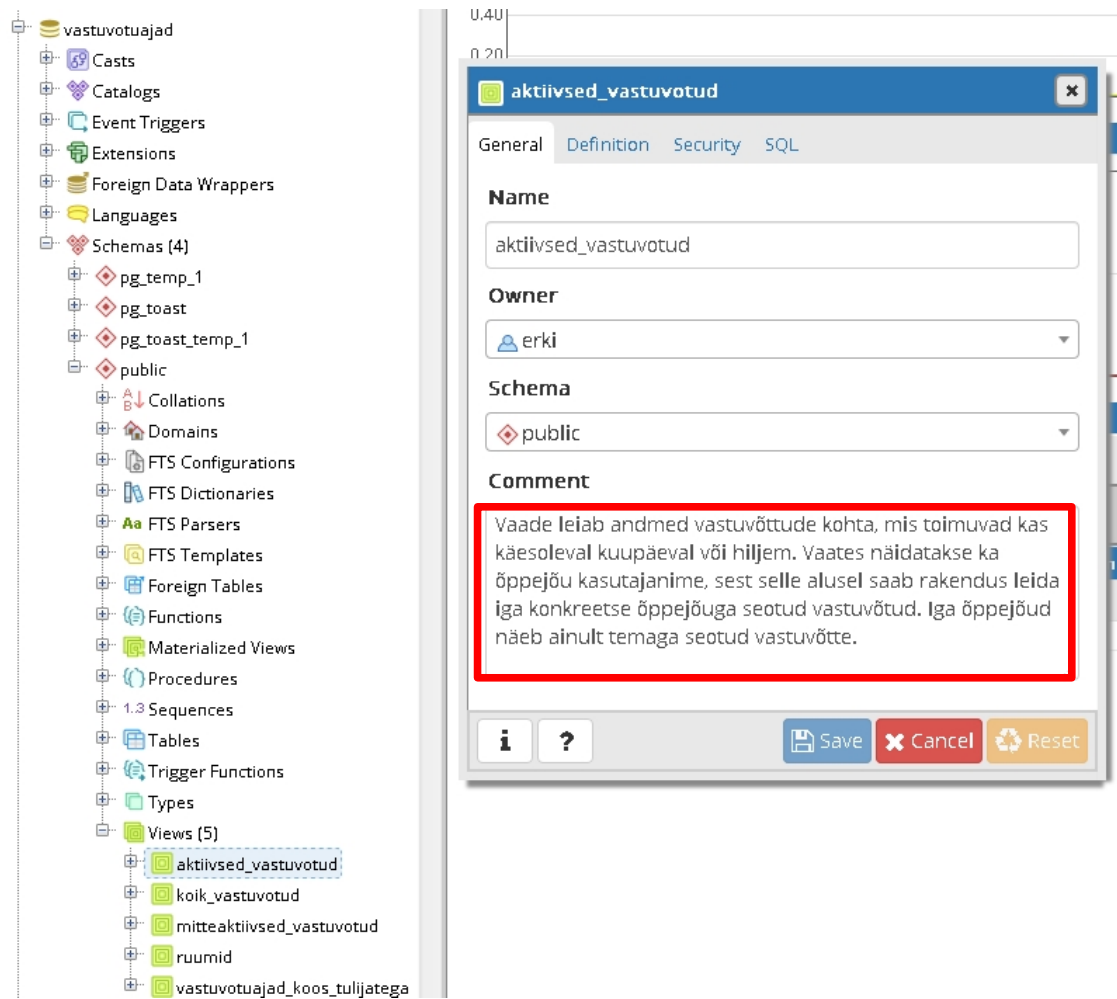
Control Source	ruum_id
Row Source	SELECT Ruum.ruum_id, Ruum.hoone_kood & '-' & Ruum.ruumi
Row Source Type	Table/Query
Bound Column	1
Limit To List	Yes
Allow Value List Edits	Yes
List Items Edit Form	

Samuti arvestage mittefunktsionaalsete nõuetega kasutajaliidese kohta (1.2.6 *Mittefunktsionaalsed nõuded*). Näiteks võib seal olla öeldud järgnevat.

- ✎ Kõikides olemite nimekirjades tuleb esitada selline hulk andmeid, et nende andmete alusel oleks võimalik olemeid üksteisest üheselt eristada ning et need andmed oleksid konkreetse kasutaja jaoks mõistetavad ja sisukad.

Kuidas näeb COMMENT lausega registreeritud kommentaare?

pgAdmin 4 (PostgreSQL)



phpPgAdmin (PostgreSQL) (toetab kuni PostgreSQL 9.6):

phpPgAdmin: PostgreSQL: vastuvotujad: public:

Tables? Views? Sequences? Functions? Full Text Search? Domains? Aggregates? Types? Operators? Op Classes? Conversions? Privileges? Export

	View	Owner	Actions				Comment
<input type="checkbox"/>	aktiivsed_vastuvotud	erki	Browse	Select	Alter	Drop	Vaade leiab andmed vastuvõttude kohta, mis toimuvad kas käesoleval kuupäeval või hiljem. Vaates näidatakse ka õppejõu kasutajanime, sest selle alusel saab rakendus leida iga konkreetse õppejõuga seotud vastuvõttud. Iga õppejõud näeb ainult temaga seotud vastuvõtte.
<input type="checkbox"/>	koik_vastuvotud	erki	Browse	Select	Alter	Drop	Vaade leiab andmed kõigi vastuvõttude kohta. Vaates näidatakse ka õppejõu kasutajanime, sest selle alusel saab rakendus leida iga konkreetse õppejõuga seotud vastuvõttud. Iga õppejõud näeb ainult temaga seotud vastuvõtte.
<input type="checkbox"/>	mitteaktiivsed_vastuvotud	erki	Browse	Select	Alter	Drop	Vaade leiab andmed vastuvõttude kohta, mis on toimunud varasemal kuupäeval kui tänane kuupäev. Vaates näidatakse ka õppejõu kasutajanime, sest selle alusel saab rakendus leida iga konkreetse õppejõuga seotud vastuvõttud. Iga õppejõud näeb ainult temaga seotud vastuvõtte.
<input type="checkbox"/>	ruumid	erki	Browse	Select	Alter	Drop	Vaade leiab andmed ülikooli ruumide ja nendega seotud hoonete kohta. Seda vaadet läheb vaja vastuvõtu registreerimisel, sest iga vastuvõtt toimub kindlas ruumis.
<input type="checkbox"/>	vastuvotujad_koos_tulijatega	erki	Browse	Select	Alter	Drop	Vaade leiab andmed kõigi vastuvõttude ja nendega seotud vastuvõtuaegade kohta. Kui vastuvõtuajaga on seotud üliõpilane, siis leiab vaade ka selle üliõpilase matricli numbrit, eesnime, perenime ja e-maili. Vaates näidatakse ka õppejõu kasutajanime, sest selle alusel saab rakendus leida iga konkreetse õppejõuga seotud vastuvõttud. Iga õppejõud näeb ainult temaga seotud vastuvõtte. Vaade on vajalik selleks, et õppejõud näeks tema vastuvõttude registreerunud üliõpilasi.

SQL Developer (Oracle):

NAITED		T990999_KORISTAMISTE_ARV				
Columns	Data	Grants	Dependencies	Details	Triggers	SQL
Actions...						
<pre>CREATE OR REPLACE FORCE EDITIONABLE VIEW "C##NAITED"."T990999_KORISTAMISTE_ARV" ("IDENT", "ARV") AS SELECT Kor.isikukood ' ' eesnimi ' ' perenimi AS ident, COUNT(Kom.isikukood) AS arv FROM t990999_Koristaja Kor LEFT JOIN t990999_Koristamine Kom ON Kor.isikukood=Kom.isikukood GROUP BY Kor.isikukood ' ' eesnimi ' ' perenimi;</pre>						
<pre>COMMENT ON TABLE "C##NAITED"."T990999_KORISTAMISTE_ARV" IS 'See vaade näitab iga koristaja kohta temaga seotud koristamiste arvu.'</pre>						

Millele vaadete loomisel tähelepanu pöörata?

- ▲ Vaadete ja nende veergude identifikaatorid e nimed.
 - Vaated ja baastabelid kuuluvad samasse nimeklassi. Ühes ja samas skeemis ei tohi olla mitu samasuguse nimega vaadet ja/või tabelit.
 - Baastabelid e tabelid ja virtuaalsed tabelid e vaated võiksid andmebaasi kasutaja seisukohast olla eristamatud. Lähtuge vaadete nimetamisel samadest põhimõtetest kui baastabelite nimetamisel. Toon näiteid.
 - Kui baastabelite puhul kasutasite nimedes *snake_case*, siis kasutage sama stiili ka vaadete puhul.
 - Kui Te ei kasuta tabelite nimede puhul ees- või järelliiteid, siis ärge tehke seda vaadete puhul. Näiteks ärge pange vaate nime algusesse "v_".
 - Kui tabelite nimed on tõstutundetud (hallatavuse seisukohalt selgelt eelistatud variant), siis peaksid seda olema ka vaadete nimed.
 - Vältimaks võimalikke probleeme programmidega, mis andmebaasi läbi vaadete kasutavad, ärge tarvitage vaadete ja nende veergude nimedes täpitähti (õõääööüü).

- Ärge kasutage nimedes üldiseid termineid nagu *andmed*, *informatsioon*, *päring* ja *alampäring*. Kõik tabelid ja vaated esitavad andmeid/informatsiooni. Kõik vaated põhinevad päringul.
 - **Halvem**: Kaupade_detail*andmed*, Kaupade_detailide_**päring**
 - **Parem**: Kaupade_detailid
- Arvestage, et Te realiseerite vaid väikest alamosa ettevõtte andmebaasist. Näiteks koondaruandeid võib olla vaja kõikide põhiolemitüüpide korral.
 - **Halb**: Koondaruanne
 - **Parem**: Kaupade_koondaruanne
- Olge nimetamisel järjekindel – näiteks kasutage vaadete nimedena järjekindlalt ja läbivalt kas ainult ainsuse vormi või ainult mitmuse vormi.
 - **Halvem**: Kauba_detailid, Aktiivsed_mitteaktiivsed_kaubad
 - **Parem**: Kaupade_detailid, Aktiivsed_mitteaktiivsed_kaubad
- Andke vaadetele sisukad nimed, mis peegeldavad nende kaudu esitatavate andmete tähendust. Kuna vaated võivad kombineerida andmeid mitmest baastabelist ja piirata neid andmeid erinevate tingimuste alusel, siis nime hea kirjeldavuse huvides võib juhtuda, et vaate nimi on pikem kui keskmine baastabeli nimi.
- Nimed peavad vastama valitud andmebaasisüsteemi reeglitele.
 - PostgreSQL: <https://www.postgresql.org/docs/11/static/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS>
 - Oracle: https://docs.oracle.com/database/121/SQLRF/sql_elements008.htm#SQLRF00223
- Kuigi vaated ja rutiinid kuuluvad erinevatesse nimeklassidesse ja seega samas skeemis võib olla sama nimega vaade ja andmebaasiserveris talletatud rutiin, siis segaduse vältimiseks ärge andke vaadetele sama nime kui on mõnel rutiinil (funktsioonil või protseduuril).
- Jälgige, et vaadete veergudel oleksid sisukad nimed. Iga vaade on tabel. Vaadete veergude nimetamisel kehtivad samad põhimõtted kui baastabelite veergude nimetamisel.

Halvem

```
SELECT T.tellimus_kood, Tl.nimetus
FROM Tellimus AS T INNER JOIN Tellimuse_liik AS Tl USING
(tellimuse_liik_kood);
```

Vaate kasutajal tekib õigustatud küsimus, kas veerus *nimetus* on tellimuste nimetused.

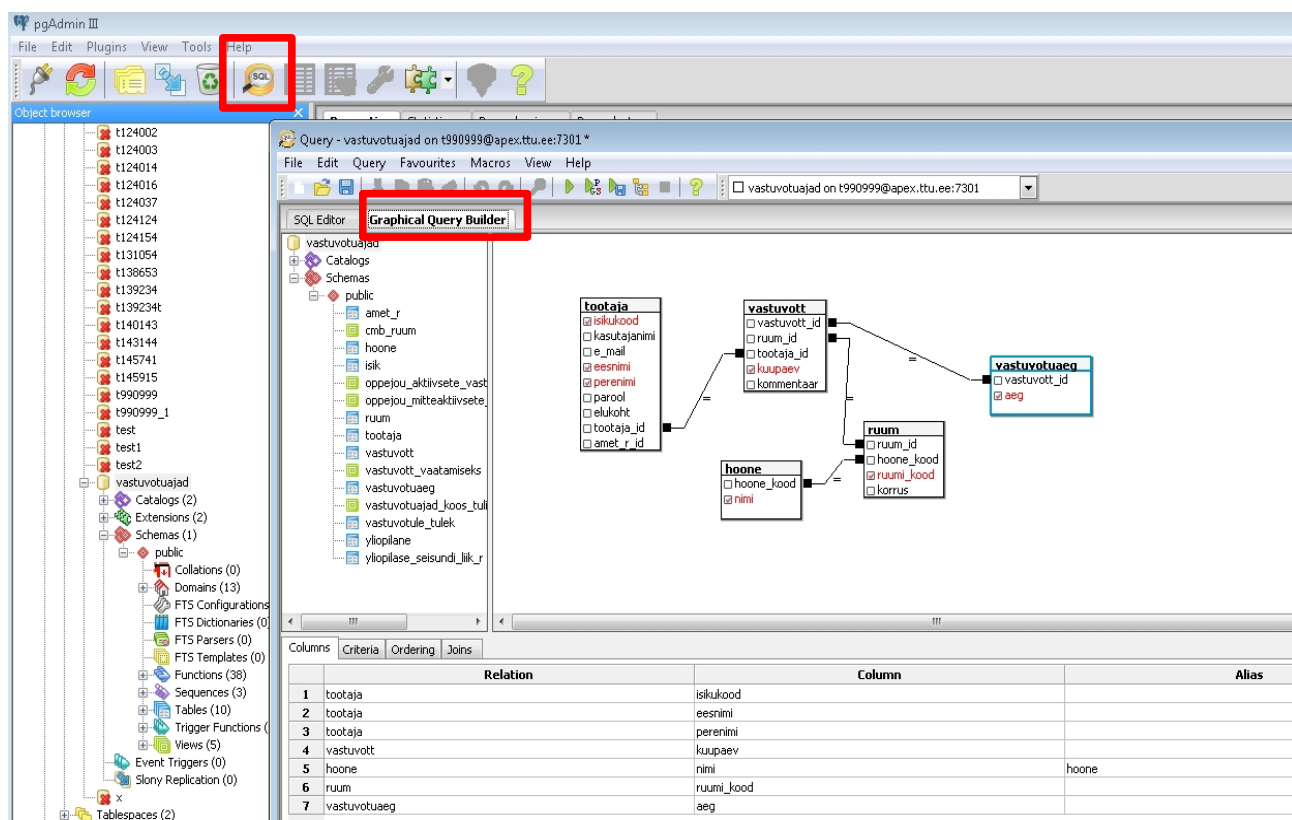
Parem

```
SELECT T.tellimus_kood, Tl.nimetus AS tellimuse_liik
FROM Tellimus AS T INNER JOIN Tellimuse_liik AS Tl USING
(tellimuse_liik_kood);
```

- Nii nagu baastabelites, vältige ka vaadete puhul veergude nimesid *id* ja *kood*, sest need on liiga üldised.
- Oracle puhul tuleb kõigis skeemiobjektide nimedes (seega ka vaadete nimedes) kasutada nimekonfliktide vältimiseks oma matrikli numbrit (nt eesliide **t990999_**). Ärge kasutage matriklinumbrit vaadete veergude nimedes.

- ⤴ Vaate loomisel kirjutage kõigepealt valmis korrektne vaate alampäring (SELECT lause). Siis pole Teil päringu tulemusena leitavate veergude hulga muutudes vaja hakata vaadet kustutama ja uuesti looma. Kui see on tehtud, siis vormistage tulemus vaateks.
- ⤴ Vaate alampäringu loomiseks võite kasutada *pgAdmin 3* (PostgreSQL), *Oracle SQL Developer* (Oracle) või *APEX* (Oracle) pakutavat graafilist SELECT lausete koostamise liidest. Kõik need liidesed sarnanevad MS Accessi *Query Designer*iga. Kõik need liidesed realiseerivad visuaalset andmebaasikeelt *Query by Example* (https://en.wikipedia.org/wiki/Query_by_Example). Olgu öeldud, et Oracle SQL Developeris ver 18 see liides toimib, kuid varasemates versioonides jooksis õppejõu kogemuste kohaselt selle kasutamisel SQL Developeri programm kohe kinni.
 - **NB!** pgAdmin 4 ei paku päringute graafilise disainimise funktsionaalsust.

pgAdmin 3 (PostgreSQL)



Pange tähele, et pgAdmin genereerib Teile tabelite ühendamisel lause, kus kasutatakse vana ühendamise süntaksi (ühendamise tingimused on WHERE klauslis). Sellisel juhul on PostgreSQL andmebaasisüsteemil vabad käed valida kõige sobivam tabelite ühendamise järjekord. Kui ühendatavaid tabeleid on kaks või kolm, ei valmista kõigi ühendamise järjekordade läbivaatamine ning parima järjekorra leidmine andmebaasisüsteemile probleemi. Kui tabeleid on rohkem, siis ei jõua süsteem enam kõiki järjekordi läbi vaadata ning võib valida mitte kõige parema ühendamise järjekorra. Päringu kirjutaja võib andmebaasisüsteemile appi tulla ja ise kõige parema järjekorra määrata, kuid see eeldab uuema ühendamise süntaksi kaustamist (ühendamise tingimused on FROM klauslis) koos parameetri *join_collapse_limit* väärtuse muutmisega. Huvi korral lugege rohkem siit:

<https://www.postgresql.org/docs/11/static/explicit-joins.html>

Oracle SQL Developer

Tuleb valida *Worksheet* saki kõrvalt sakk *Query Builder*.

Output	Expression	Aggregate	Alias	Sort Type	Sort Order	Grouping	Criteria	Or
<input checked="" type="checkbox"/>	C##TUD1.T990999_KORISTAJA.ISIKUKOOD			Ascending	2	<input type="checkbox"/>		
<input checked="" type="checkbox"/>	C##TUD1.T990999_KORISTAJA.EESNIMI					<input type="checkbox"/>		
<input checked="" type="checkbox"/>	C##TUD1.T990999_KORISTAJA.PERENIMI					<input type="checkbox"/>		
<input checked="" type="checkbox"/>	C##TUD1.T990999_KORISTAMINE.RUUMI_NUMBER					<input type="checkbox"/>		
<input checked="" type="checkbox"/>	C##TUD1.T990999_KORISTAMINE.ALGUSE_AEG			Ascending	1	<input type="checkbox"/>		
<input checked="" type="checkbox"/>	C##TUD1.T990999_KORISTAMINE.LOPU_AEG					<input type="checkbox"/>		

Töõlehel (*Worksheet*) on näha genereeritud SQL lause.

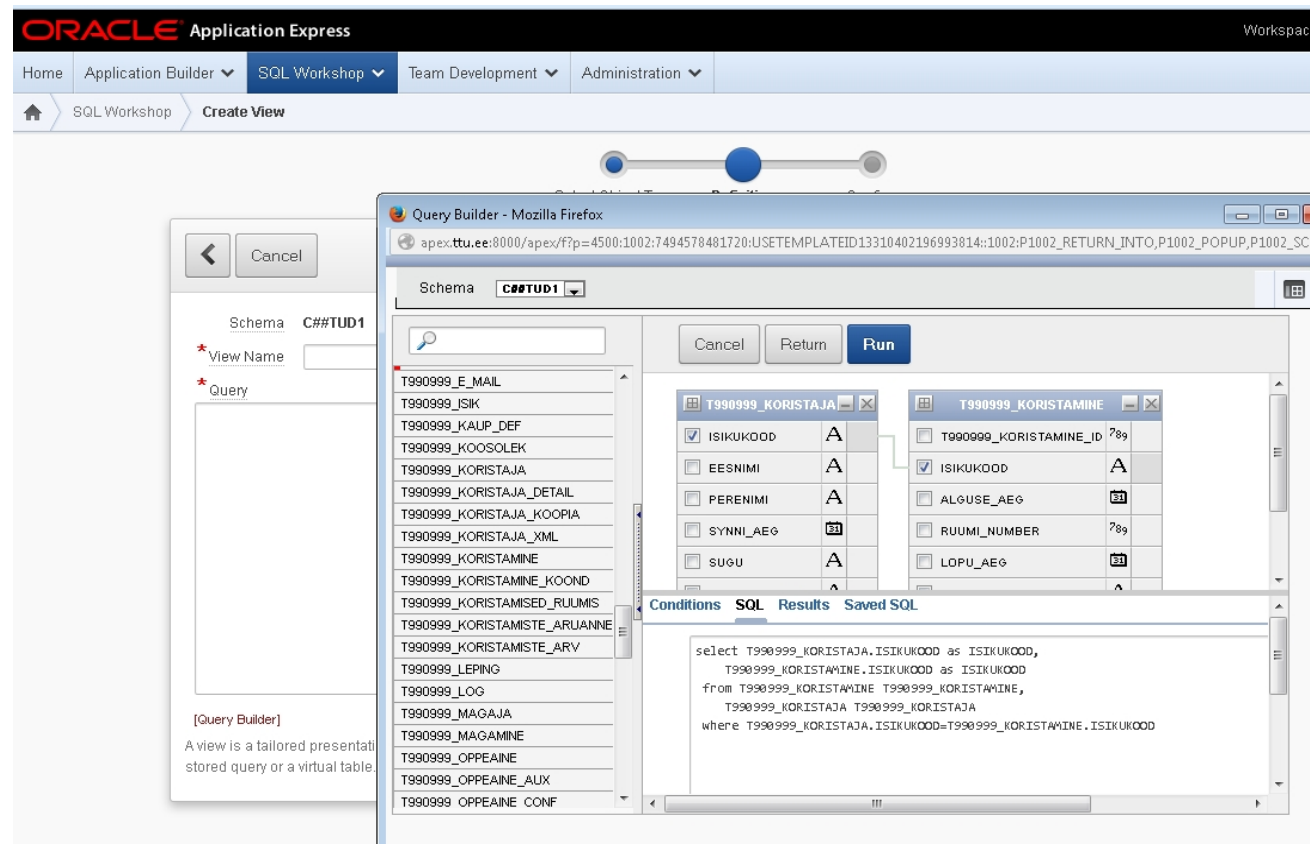
```

SELECT
  c##tudl.t990999_koristaja.isikukood,
  c##tudl.t990999_koristaja.eesnimi,
  c##tudl.t990999_koristaja.perenimi,
  c##tudl.t990999_koristamine.ruumi_number,
  c##tudl.t990999_koristamine.alguse_aeg,
  c##tudl.t990999_koristamine.lopu_aeg
FROM
  c##tudl.t990999_koristaja
  INNER JOIN c##tudl.t990999_koristamine ON c##tudl.t990999_koristaja.isikukood = c##tudl.t990999_koristamine.isikukood
ORDER BY
  c##tudl.t990999_koristamine.alguse_aeg,
  c##tudl.t990999_koristaja.isikukood

```


Oracle APEX

Tuleb valida: SQL Workshop => Object Browser => Create => View => Query Builder



- ⚠ Ärge kasutage vaadete alampäringutes **SELECT *** ..., vaid kirjutage välja veerud, mida soovite vaates näha. Näide: **SELECT isikukood, eesnimi, perenimi** Sellega dokumenteerite/määrate üheselt vaates soovitud veerud. Vaate aluseks olevates baastabelites võib aja jooksul veergude hulk muutuda – sama **SELECT *** vaate loomise lause käivitamine eri aegadel võib anda tulemuseks erineva struktuuriga vaate.
- ⚠ Kui loote lõppkasutajatele andmete esitamiseks mõeldud vaateid, siis näidake kasutajatele välisvõtmeteks olevate ID-de asemel (ning mõnikord ka koodide asemel – sõltub sellest kui hästi oodatavad kasutajad neid koodi tunnevad) **kasutajatele tähendust omavaid andmeid**. Pange tähele, et järgnevas näites leitakse lisaks töötajate ees- ja perenimedele ka töötajate meiliaadressid, sest mitmel erineval töötajal võib olla ühesugune ees- ja perenimi ning meiliaadress aitab neid töötajaid üksteisest eristada.

Halvem

```
SELECT vastuvott_id, tootaja_id, toimumise_kuupaev
FROM Vastuvott;
```

Parem

```
SELECT vastuvott_id, '(' || e_mail || ')' ' ||
Coalesce(eesnimi,'') || ' ' || Coalesce(perenimi,'') AS
oppejoud, kuupaev
FROM Vastuvott INNER JOIN Tootaja USING(tootaja_id);
```

- ⤴ See vaade on mõeldud kasutajale, kes tahab ülevaadet *kõigi õppejõudude* vastuvõttudest.
- ⤴ Selleks, et iga õppejõud saaks vaadata ainult enda vastuvõttude andmeid, oleks vaja luua teine vaade. Selles pole mõtet esitada meiliaadressi, eesnime ja perenime (õppejõud teab oma nime ja meiliaadressi), küll aga oleks seal vaja esitada töötaja kasutajanimi (tõsi küll, sellena võib olla kasutusel meiliaadress). Kui töötaja logib süsteemi sisse ja tuvastatakse, siis kasutajanime alusel leiab rakendus read, mida igale konkreetsele töötajale kuvada.
- ⤴ Kui ühendate mittekohustuslikes veergudes olevaid stringe (nagu *eesnimi* ja *perenimi*), siis peate teadma, kuidas konkreetsetes andmebaasisüsteemis käitub stringide ühendamise operaator, kui üks argument on NULL.
 - *MS Accessis*: NULL + 'väärtus' => NULL
 - *MS Accessis*: NULL & 'väärtus' => 'väärtus'
 - *PostgreSQLis*: NULL || 'väärtus' => NULL
 - *Oracles*: NULL || 'väärtus' => 'väärtus' (Oracle asendab NULLi automaatselt tühja stringiga)
- ⤴ PostgreSQLis kasutan süsteemi-defineeritud funktsiooni *Coalesce*, et asendada NULL tühja stringiga.
 - *PostgreSQLis*: " || 'väärtus' => 'väärtus'
 - <https://www.postgresql.org/docs/11/static/functions-conditional.html#FUNCTIONS-COALESCE-NVL-IFNULL>
- ⤴ Kui tõlgite MS Accessi päringuid sobivaks PostgreSQL või Oracle jaoks, siis tuleb tabelite ja veergude nimede ümbert eemaldada seal võimalik et olevad nurksulud. Nurksulud on mõeldud kasutamiseks MS Accessi ja MS SQL Serveri SQL mägimurrakus ja tähistavad seal piiritletud identifikaatorit. PostgreSQLis ja Oracles (ning ka SQL standardi kohaselt) on piiritletud identifikaatorid jutumärkides. Piiritletud identifikaator võib sisaldada tühikut või olla SQLi võtmesõna; piiritletud identifikaator on tõstutundlik jms.

Vale

```
SELECT [vastuvott_id], [tootaja_id], [toimumise_kuupaev]
FROM [Vastuvott];
```

Õige

```
SELECT vastuvott_id, tootaja_id, toimumise_kuupaev
FROM Vastuvott;
```

- ⤴ MS Accessi *if* funktsioonile vastab PostgreSQLis ja Oracles CASE avaldis.
 - <https://www.postgresql.org/docs/11/static/functions-conditional.html#FUNCTIONS-CASE>
 - <https://docs.oracle.com/database/121/SQLRF/expressions004.htm>

- ⤴ Kui vaates olevaid ridu on vaja piirata klassifikaatori väärtuse alusel, siis saate tingimuses kasutada klassifikaatori koodi ning pole vaja viia läbi ühendamisoperatsiooni klassifikaatori tabeliga, et leida sealt koodile vastav nimetus.

Halvem

```
SELECT T.tellimus_kood, TS.nimetus AS tellimuse_seisund
FROM Tellimus AS T INNER JOIN Tellimuse_seisundi_liik AS
TS USING (tellimuse_seisundi_liik_kood)
WHERE TS.nimetus='aktiivne';
```

- ⌘ Leiate ainult ühes kindlas seisundis olevad tellimused – milleks on vaja igas reas korrata (antud juhul ühte ja sama) seisundi nimetust?
- ⌘ Kui olete dokumenteerinud, et nt seisundile "aktiivne" vastab kood 2, siis miks Te ei otsi tellimust selle koodi alusel?
- ⌘ Kui soovite päringu tulemuses kangesti näha sõnalist seisundi kirjeldust, siis võiks tellimuse seisundi liigi kood olla andmebaasis tekstiline (näiteväärtused LOOD, KINNIT, AKT, TÜHIST, ARHIV). Sellisel juhul saaksite päringus väljastada selle koodi, kuid ikkagi pole vaja päringus ühendada tabelit *Tellimuse_seisundi_liik*.

Parem

```
SELECT tellimus_kood
FROM Tellimus
WHERE tellimuse_seisundi_liik_kood=2;
```

Tabelite *Tellimus* ja *Tellimuse_seisundi_liik* ühendamine on õigustatud, kui on täidetud *mõlemad* järgmised tingimused.

- ⌘ Vaate alampäringuga soovitakse leida *mitmes erinevas* seisundis olevate tellimuste ühend.
- ⌘ Vaade on mõeldud lõppkasutajale andmete esitamiseks ning kasutaja vajab infot kaupade seisundi kohta (lõppkasutajale ütleb seisundi nimetus ilmselt rohkem kui kood).

```
SELECT T.tellimus_kood, TS.nimetus AS tellimuse_seisund
FROM Tellimus AS T INNER JOIN Tellimuse_seisundi_liik AS
TS USING (tellimuse_seisundi_liik_kood)
WHERE T.tellimuse_seisundi_liik_kood IN (1,2,3);
```

- ⌘ "Andmebaasid I" projektis ei pidanud Te prototüübis realiseerima kasutaja tuvastamist. Jätsite prototüübi kasutajale mulje, et ta on konkreetse kasutajana sisse loginud. Mõned loodud päringud võisid olla näiteks sellised:

Näide:

```
SELECT tellimus_kood, tahtaeg
FROM Tellimus
WHERE klient_id=1;
```

"Andmebaasid II" projektis tuleb Teil realiseerida kasutaja tuvastamine, kui see on loogiliselt vajalik. Sisseloginud kasutaja näeb ainult temaga seotud/talle määratud andmeid. Vaadete alampäringute tingimused ei saa enam piirata ridade hulka konkreetse kasutaja järgi. Eelnevalt nimetatud vaate alampäring võiks olla hoopis järgnev.

```
SELECT tellimus_kood, tahtaeg, Isik.kasutajanimi AS klient
FROM Tellimus INNER JOIN Isik ON
Tellimus.klient_id=Isik.isik_id;
```

Vaates leitakse iga tellimuse puhul selle esitanud kliendi kasutajanimi. Kui klient logib sisse, siis programm jätab kasutajanime meelde. Kui klient tahab vaadata oma tellimusi, siis programm teeb selle vaate põhjal päringu ning leiab kasutajanime järgi üles ainult päringu käivitanud kliendi tellimused.

Sellised päringud on tõenäolisemad, kui realiseerite kliendi töökohta. Üldjuhul ei tohiks ükski klient näha teiste klientidega seotud andmeid.

X halduri (kus X on kaup, teenus, parklakoht, treeningu kirjeldus vms) töökohal saavad haldurid hallata ilmselt kõiki X eksemplare, sõltumata sellest, milline haldur nende andmed sisestas (see sõltub ettevõtte töökorraldusest). Juhatajatel peab olema ülevaade kõigist X eksemplaridest, muidu nad oma tööd teha ei saaks.

- ▲ Pöörake vaadete parema kasutatavuse huvides tähelepanu veergude järjekorrale. Esitage vaates veerud järjekorras, kus kõigepealt on veerud, milles olevad väärtused võiksid olla vaates olevate ridade unikaalsed identifikaatorid (nt isikuandmete vaates, isikukood ja selle väljastanud riigi nimi või meiliaadress). Veergude nimekirja lõpus on kõigepealt mittekohustuslikke andmeid sisaldavad veerud (nt isikuandmete vaates isiku sünniaeg ja kommentaar) ning kõige lõpus veerud, milles olevad andmed esitavad metaandmeid (nt isikuandmete registreerija ja registreerimise_aeg).

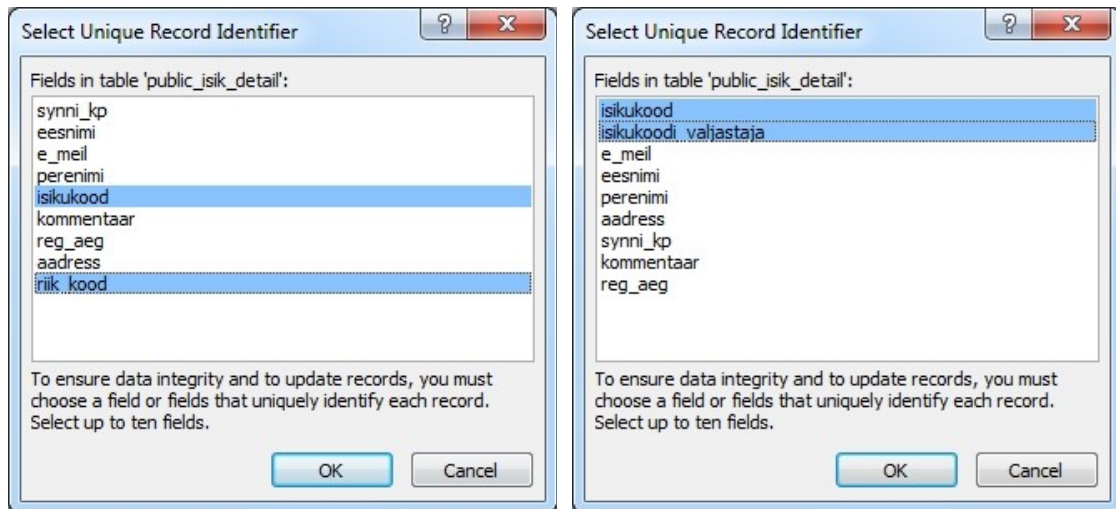
Halvem

```
SELECT synni_kp, eesnimi, e_meil, perenimi, isikukood,
kommentaar, reg_aeg, aadress, riik_kood
FROM Isik;
```

Parem

```
SELECT isikukood, riik_kood AS isikukoodi_valjastaja,
e_meil, eesnimi, perenimi, aadress, synni_kp,
kommentaar, reg_aeg
FROM Isik;
```

Näiteks, on siis mugavam MS Accessi andmebaasi serveri andmebaasist vaate linkimisel määrata selle unikaalset identifikaatorit. Võrrelge:



- ⚡ Oletame, et registreerite andmebaasis kasutajate kasutajanimed/paroolid. Vaadetes, mille kaudu näitate kasutajatele kasutajate nimekirja, pole mingit põhjust esitada parooli e salasõna ja soola. Parool on andmebaasis räsitud ja soolatud ning lõppkasutajal pole sellega midagi peale hakata. See võiks huvi pakkuda ainult ründajale. Parooli/soola esitamine vaates tuleb kõne alla, kui seda vaadet kasutab kasutaja tuvastamiseks mõeldud rutiin.
- ⚡ Eelistage väiksemaid ja konkreetse probleemi lahendamisele mõeldud vaateid suurematele ja üldistele vaadetele. Vältige vaadete loomist vaadete põhjal (muudavad vaated raskemini hallatavaks ja arusaadavaks). Üldised ja suured vaated (palju veerge, ilmselt küsitakse andmeid paljudest tabelitest) on vastuolus otstarbe lahususe (*separation of concerns*) disainiprintsiibiga, mille idee vastavalt peab igal tarkvara elemendil (sh vaatel) olema üks kindel ülesanne. Objektorienteeritud koodis iseloomustavad sarnast probleemi halvad lõhnad:

 - *Plekk e Jumal-klass*: <https://sourcemaking.com/antipatterns/the-blob>
 - *Suur klass*: <https://refactoring.guru/smells/large-class>
- ⚡ MS Accessi prototüübis võisite keerukamate päringute realiseerimiseks kasutada stiili, et jagasite lahenduse mitmeks osaks, realiseerisite need eraldi nimeliste päringutena (vaadetena) ja panite nende põhjal kokku lõpptulemuse. Selline lahendus on põhimõtteliselt võimalik ka PostgreSQLis ja Oracles, kus alamülesannete lahendamiseks saaks luua CREATE VIEW lausega vaated ning vaadete põhjal saab omakorda luua uusi vaateid. Probleemiks on, et suures andmebaasis tekiks vaateid väga palju ja tuleks hakata looma vaateid vaadetele. Sellise olukorra haldamine kasvab kiiresti üle pea ja tulemuseks on näiteks vaadete dubleerimine – loote alampäringu poolest identseid või peaaegu identseid, kuid erineva nimega vaateid. **Alternatiiviks** on kasutada vaate alampäringus WITH klauslit. Seda kirjeldab nii SQL standardi hetkel kehtiv versioon kui saate seda ka kasutada PostgreSQLis ning Oracles. WITH klausel võimaldab päringu alamosana kirjeldada nimelise alampäringu (manustatud vaade, *inline view*), millele saab viidata ainult selle päringu piires. Igas WITH klausli alampäringus saab viidata ka kõigile selles klauslis eelnevalt defineeritud alampäringutele. Järgnevat näidet saate katsetada nii PostgreSQLis

(**apex.ttu.ee** serveri andmebaas *scott*), kui Oracles. Ülesandeks on leida osakondade arv, kus töötab vähemalt üks töötaja ametikohaga "CLERK".

```
WITH emp_clerk AS (SELECT empno, deptno FROM Emp WHERE
job='CLERK'),
diff_dept AS (SELECT DISTINCT deptno FROM emp_clerk)
SELECT Count(deptno) AS cnt
FROM diff_dept;
```

vs.

```
SELECT Count(DISTINCT deptno) AS cnt
FROM (SELECT empno, deptno FROM Emp WHERE job='CLERK')
emp_clerk;
```

vs.

```
SELECT Count(deptno) AS cnt
FROM (SELECT DISTINCT deptno
FROM (SELECT empno, deptno FROM Emp WHERE job='CLERK')
emp_clerk) diff_dept;
```

WITH klausli kasutamise eelis võrreldes alampäringute kirjutamisega FROM klauslisse on:

- ⤴ alampäringu taaskasutamise võimalus samas lauses (st alampäringule võib viidata lauses rohkem kui ühest kohast),
- ⤴ parem loetavus – lahenduskäik on lineaarselt jälgitav, mitte ei pea erinevate klauslite vahel pilguga edasi-tagasi hüplema.
- ⤴ Vältige PostgreSQLis päringutes (sh vaadetes) NOT IN või <>ALL predikaadi koos *mittekorreleeruva* alampäringuga kasutamist, sest selline päring täidetakse suurte andmehulkade korral aeglaselt.

Näide:

Tabelis *Person* (Isik) on 70000 rida

Tabelis *Health_care_visit* (Tervishoiuasutuse külastus) on 1000000 rida

Ülesandeks on leida selliste isikute arv, kellel ei ole ühtegi tervishoiuasutuse külastust.

```
SELECT Count(*) AS cnt FROM person WHERE party_id NOT IN
(SELECT patient_id FROM Health_care_visit);
```

apex.ttu.ee serveris võttis selle päringu täitmise aega järgnevalt.

- ⤴ PostgreSQL 9.5 korral **45 minutit**.
- ⤴ PostgreSQL 10 korral **22 minutit**.
- ⤴ PostgreSQL 11 korral **5 minutit ja 30 sekundit**.

Päringu

```
SELECT Count(*) AS cnt FROM person WHERE party_id <> ALL
(SELECT patient_id FROM Health_care_visit);
```

täitmine võttis PostgreSQL 10 korral **21.5 minutit** ja PostgreSQL 11 korra **5 minutit ja 30 sekundit**.

Korreleeruvat alampäringut sisaldava päringu

```
SELECT Count(*) AS cnt FROM person AS p WHERE NOT EXISTS
(SELECT 1 FROM Health_care_visit AS hcv WHERE
p.party_id=hcv.patient_id);
```

täitmine võttis PostgreSQL 10 korral **0.15 sekundit** ja PostgreSQL 11 korral **0.25 sekundit**.

- ✦ Kasutage vaadete alampäringutes konkreetse andmebaasisüsteemi andmekäitluskeelee kõiki võimalusi. Praktiliselt kõik, mida õppisite SELECT lausete kohta MS Accessi baasil, toimib ka PostgreSQLis ja Oracles. Küll aga pakuvad need süsteemid võrreldes MS Accessiga päringute kirjutajatele *väga palju* täiendavaid võimalusi. Olen kindel, et hämmeldute, kui palju asju saab SQL andmekäitluskeelega ära teha. Huvi korral lugege palun lisamaterjalide lehelt slaidikomplekti **SQL_DML_alamkeelee_voimalusi_IDU0230_2018.ppt**.

Kindlasti soovitan uurida kokkuvõttefunktsioone *listagg* (Oracle) ja *string_agg* (PostgreSQL). Vaadake nende kohta eelnimetatud slaidikomplektist slaide. Näiteks kui tahate vaate kaudu näidata andmeid mingit tüüpi põhiobjektide (nt *Kaup*) ja nendest sõltuvate objektide (nt *Kauba kategooria omamine*) kohta, siis saate info sõltuvate objektide kohta esitada vaates tekstistringina (nt ühes veerus kauba nimi ning teises veerus kauba kategooriate komadega eraldatud nimekiri), mida on mugav lõppkasutajale serveerida. Samuti saate vaates esitada soovi korral andmeid massiividenä või JSON dokumentidenä, mille vaata alampäring konstrueerib käigupealt baastabelites olevate andmete põhjal. Lisamaterjalide lehelt leiate ka selle kohta näiteid.

- ✦ Kasutage PostgreSQL vaadetes turvabarjääri (*security_barrier*). See tagab, et vaatesse ei saa "sisse häkkida", nägemaks andmeid, mida vaate looja poolt ei soovita vaate kasutajatele näidata.
- ✦ Kasutage vaadetes, mille kaudu on *põhimõtteliselt* võimalik muuta andmebaasis olevaid andmeid, WITH CHECK OPTION kitsendust. Sellega tagate, et vaate kaudu saab teha vaid andmemuudatusi, mis on kooskõlas vaate alampäringu tingimustega. PostgreSQLi lisandus selle määramise võimalus alates PostgreSQL 9.4, Oracles oli juba varem olemas.
- ✦ Kui teate Oracle vaate korral, mille kaudu *teoreetiliselt* on võimalik andmeid muuta, et ei plaani seda kunagi teha, siis deklareerige vaates WITH READ ONLY kitsendus. PostgreSQLis tuleks samasuguse funktsionaalsuse

saavutamiseks kirjutada rohkem koodi, luues vaatega seotud **INSTEAD OF** trigereid või **DO INSTEAD** reegleid.

Järgnev päring *PostgreSQL* andmebaasi süsteemikataloogi põhjal võimaldab teha kindlaks, milliste kasutaja loodud vaadete kaudu saab *PostgreSQL* andmebaasis *põhimõtteliselt* andmeid muuta. Kõik sellised vaated tuleb luua **WITH CHECK OPTION** kitsendusega.

```
SELECT table_schema AS schema, table_name AS view,
is_updatable, is_insertable_into
FROM Information_schema.views
WHERE (is_insertable_into='YES' OR is_updatable='YES')
AND table_schema NOT IN (SELECT schema_name
FROM INFORMATION_SCHEMA.schemata
WHERE schema_name<>'public' AND
schema_owner='postgres' AND schema_name IS NOT NULL)
ORDER BY table_schema, table_name;
```

Järgnev päring *Oracle* andmebaasi süsteemikataloogi põhjal võimaldab teha kindlaks, milliste kasutaja loodud vaadete kaudu saab *Oracle* andmebaasis *põhimõtteliselt* andmeid muuta. Kõik sellised vaated tuleb luua kas **WITH CHECK OPTION** või **WITH READ ONLY** kitsendusega. Asendage **990999** päringus enda matrikli numbriga.

```
SELECT DISTINCT table_name AS updatable_view
FROM user_updatable_columns
WHERE table_name LIKE '%990999%'
AND table_name NOT IN (SELECT tname FROM Tab WHERE
tabtype='TABLE')
AND (updatable='YES' OR insertable='YES' OR
deletable='YES')
ORDER BY table_name;
```

- ⤴ Vaade (virtuaalne tabel) on tabel. *Oracle* võimaldab deklareerida vaadetele **PRIMARY KEY**, **UNIQUE** ja **FOREIGN KEY** kitsendusi. Neid kitsendusi andmebaasis ei jõustata (kitsendused peab looma **DISABLE NOVALIDATE** seisundis). Samas võib seda infot kasutada andmebaasisüsteemi optimeerimismoodul vaadete põhjal tehtavate päringute kiiremaks täitmiseks. Selleks peaks need kitsendused looma **RELY** omadusega – **RELY DISABLE NOVALIDATE** režiimis kitsendus tähendab, et kuigi andmebaasisüsteem kitsenduse täidetust ei jõusta/kontrolli, öeldakse andmebaasisüsteemile, et usutavasti see kitsendus kehtib. Seda infot saavad ka kasutada arendajad ning andmebaasi süsteemikataloogist andmeid lugevad rakendused. *PostgreSQL*is ei ole võimalik vaadetele **PRIMARY KEY**, **UNIQUE** ja **FOREIGN KEY** kitsendusi defineerida.
 - <http://dbaora.com/constraints-on-views-in-oracle/>
 - <https://stackoverflow.com/questions/4435034/whats-the-point-of-a-view-constraint>
 - <https://stackoverflow.com/questions/29180368/rely-constraint-in-view>
- ⤴ Üritage mitte luua vaateid, mis tükeldavad horisontaalselt mingis baastabelis olevaid andmeid (v.a põhiobjekti seisundi alusel), kusjuures see

tükeldus võib aja jooksul suure tõenäosusega täieneda (<http://it-ebooks.info/book/3451/> Peatükk 9). Näiteks esitate parklakohtade tabeli põhjal eraldi vaated esimesel, teisel ja kolmandal korrusel olevate parklakohtade kohta. Kui parklale ehitatakse peale neljas korrus, siis tuleb hakata infosüsteemi tarkvara täiendama. Selliste vaadete asemel võite luua tabelifunktsiooni, millele antakse väljakutsel argumendina ette korruse number (vt järgmine punkt).

- ⌘ Vaated on virtuaalsed tabelid. Baastabelite korral oli oluline andmete liiasuse vähendamine (vt normaliseerimine, ortogonaalse andmebaasi disaini printsiip). Kuidas on lood vaadetega? Vaadete väärtused arvutatakse andmebaasisüsteemi poolt välja baastabelite väärtuste alusel ja muutuvad kohe kui muutuvad andmed baastabelites. Andmemuudatused tehakse lõppkokkuvõttes ikkagi baastabelites, kuigi vaated võivad olla muudatustele vahendajateks. Selles mõttes ei ole see probleem, kui vaadete normaliseerituse aste on madalam kui viies normaalkuju ja samu andmeid korraldatakse erinevates vaadetes. Tegemist on *kontrollitud andmete liiasusega*. Vaated peavad serveerima kasutajatele andmeid sellisel kujul nagu nad soovivad ja võibolla mõni kasutaja eelistab sellist liiasust. Siin on tegemist "Ära korda ennast" disainiprintsiibi (<http://wiki.c2.com/?DontRepeatYourself>) ilminguga. Andmetest on süsteemis üks, ühemõtteline ja autoriteetne esitus – esitus baastabelites ja vaated ainult kasutavad neid andmeid, mitte ei hoia neid ise. Siiski tuleb parema hallatavuse huvides vältida järgnevat.

 - Ärge dubleerige vaateid – ärge looge erineva nimega (või erinevates skeemides sama nimega) vaateid, millel on ühte ja sama ülesannet lahendav alampäring. Pidage meeles, et SQLis saab ühte ja sama ülesannet lahendada enamasti paljudel erinevatel viisidel ja seega võib lausete kirjpilt (süntaks) olla vägagi erinev, kuid need lahendavad ikkagi sama ülesannet. Teisalt võib erinevus olla nii väike nagu veergude järjekord päringu tulemuseks olevas tabelis. Veergude järjekord ei muuda andmete tähendust ja erineva veergude järjekorra jaoks eraldi vaateid looma hakata pole mingit põhjust.
 - Õigustus, et need erinevad vaated on mõeldud erinevat tüüpi kasutajatele ei pea vett, sest juurdepääsu reguleerimiseks on õiguste mehhanism. Selleks ei ole vaja vaadete koodi dubleerida ja tuua endale kaela kõiki sellest tulenevaid hädasid.
- ⌘ Vaadete (ja ka hetktõmmiste) alampäringutes võite pöörduda nii süsteemi-defineeritud (nt *Upper*, *string_agg*) kui ka kasutaja-defineeritud funktsioonide poole. PostgreSQLis vältige võimalusel pöördumist *plpgsql* keeles loodud kasutaja-defineeritud funktsioonide poole. Suure tõenäosusega saab sellise funktsiooni realiseerida SQL keeles. SQL funktsiooni poole pöördumine on jõudluse poolest eelistatum, sest ei tekita kontekstkommutatsiooni (ingl *context switch*) (vt "Andmebaasid II" teema 4 materjale).
- ⌘ Vaate asemel võib ka luua tabelifunktsiooni. Tabelifunktsioon tagastab ridade hulga. Kasutage tabelifunktsiooni eeskätt siis, kui soovite juhtida funktsiooni tagastatavate ridade hulka funktsioonile ette antavate argumentidega. Lugege huvi korral tabelifunktsioonide kohta lisamaterjalide lehel olevast failist

PostgreSQL_operaatorid_SQL_funktsioonid_IDU0230_2018.ppt

Sealt leiate ka ühe näite Oracle kohta.

- ✧ Kui soovite parandada päringute töökiirust, siis võite vaate asemel luua hetktõmmise e materialiseeritud vaate. Need on näide kontrollitud andmete liiasusest, st andmete lugejal ja kirjutajal ei ole endal vaja muretseda hetktõmmise värskendamise pärast. Seda teeb süsteem ise. Enamasti seadistatakse värsekendamine toimuma viiteajaga (nt kord päevas) ja sellist värskendamist kutsutakse *asünkroonseks*.
 - Nimetamisel ja liiasuse vältimisel lähtuge samadest põhimõtetest kui vaadete puhul.
 - Kuna hetktõmmises on koopia andmetest, siis suurendavad nad andmete salvestusruumi vajadust. Neid tuleks luua vajaduspõhiselt – aeglaselt täidetavate päringute kiirendamiseks, mitte "lennukilt külvates" kõik vaated hetktõmmistega asendada.
 - Andmemuudatuste kiirendamiseks eelistage asünkroonset värskendamist sünkroonsele, kuid mõelge läbi värskendamise aeg ja sagedus (võiks toimuda siis, kus süsteemi koormus on madalam).
 - Oracles on värskendamise automatiseerimise taristu andmebaasisüsteemi sisseehitatud (sagedus ja ulatus kirjeldatakse CREATE MATERIALIZED VIEW lauses), Unix/Linux operatsioonisüsteemil töötava PostgreSQL'i korral tuleb näiteks kasutada *cron* utiliiti. Selle abil määratakse tööde ajakava töötlemiseks tulevikus. Hetktõmmise värskendamise töö sisuks on REFRESH MATERIALIZED VIEW lausete käivitamine. Oracles saab värskendamist erakorraliselt algatada, kasutades süsteemi-defineeritud paketi DBMS_MVIEW olevaid rutiine.
 - <https://stackoverflow.com/questions/29437650/how-can-i-ensure-that-a-materialized-view-is-always-up-to-date>

PostgreSQL – CREATE MATERIALIZED VIEW

<https://www.postgresql.org/docs/11/static/sql-creatematerializedview.html>

Oracle – CREATE MATERIALIZED VIEW

https://docs.oracle.com/database/121/SQLRF/statements_6002.htm

Oracle – CREATE MATERIALIZED VIEW LOG (vajalik, kui soovite ainult muutunud osa värskendamist, mitte kogu vaate väärtuse nullist uuesti arvutamist)

https://docs.oracle.com/database/121/SQLRF/statements_6003.htm

PostgreSQL – REFRESH MATERIALIZED VIEW

<https://www.postgresql.org/docs/11/static/sql-refreshmaterializedview.html>

PostgreSQL – DROP MATERIALIZED VIEW

<https://www.postgresql.org/docs/11/static/sql-dropmaterializedview.html>

Oracle – DROP MATERIALIZED VIEW

https://docs.oracle.com/database/121/SQLRF/statements_8022.htm

Oracle – DROP MATERIALIZED VIEW LOG

https://docs.oracle.com/database/121/SQLRF/statements_8023.htm

- ⤴ Järgnev punkt puudutab nii vaateid kui ka andmebaasiserveris talletatud rutiine (ülesanne 8). Ärge asendage päringutes (sh vaadete alampäringutes) tabelite ühendamist funktsiooni väljakutsega. See vähendab päringute täitmise kiirus. Tegin apex.ttu.ee serveris PostgreSQL (10) korral järgmise katsetuse.

Facility (tervishoiuasutus): 50 000 rida

Hcv (arsti vastuvõtt, mis toimub tervishoiuasutuses): 1 000 000 rida

```
SELECT f.facility_id, facility_name,
Count(hcv.facility_id) AS cnt
FROM Facility AS f LEFT JOIN Hcv ON
f.facility_id=hcv.facility_id
GROUP BY f.facility_id, facility_name
ORDER BY cnt DESC;
```

Täitmisplaani koostamiseks + täitmiseks kulus: **907 ms**

```
CREATE OR REPLACE FUNCTION f_hcv_sql (p_facility_id
Facility.facility_id%TYPE) RETURNS BIGINT AS
$$
SELECT Count(*) AS cnt FROM hcv WHERE
facility_id=p_facility_id;
$$ LANGUAGE sql
SECURITY DEFINER
SET search_path=public, pg_temp;

SELECT facility_id, facility_name, f_hcv_sql(facility_id)
AS cnt
FROM Facility
GROUP BY facility_id, facility_name
ORDER BY cnt DESC;
```

Täitmisplaani koostamiseks + täitmiseks kulus: **1819 ms**

```
CREATE OR REPLACE FUNCTION f_hcv_plpgsql (p_facility_id
Facility.facility_id%TYPE) RETURNS BIGINT AS
$$
DECLARE
cnt BIGINT;
BEGIN
SELECT Count(*) INTO cnt FROM hcv WHERE
facility_id=p_facility_id;
RETURN cnt;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER
SET search_path=public, pg_temp;

SELECT facility_id, facility_name,
f_hcv_plpgsql(facility_id) AS cnt
FROM Facility
GROUP BY facility_id, facility_name
ORDER BY cnt DESC;
```

Täitmisplaani koostamiseks + täitmiseks kulus: **2560 ms**

Kõige rohkem aega kulus PL/pgSQL funktsiooni kasutamise korral (peaaegu kolm korda rohkem kui ühendamist sisaldava SQL lause korral) – põhjuseks kontekstkommutatsioon. Kuid ka sellisel juhul kui tabelite ühendamine oli asendatud SQL funktsiooni väljakutsega oli päringu täitmiseks kulunud aeg kaks korda suurem kui tabelite ühendamist kasutavas päringus. Seega tuleks antud juhul ka SQL funktsiooni kasutamisest loobuda.

Kordasin sama katsetust PostgreSQL 11 ning tulemused olid üllatavad.

- ⤴ Kõikide lausete täitmise aeg oli kordade aeglasem kui PostgreSQL 10 korral.
- ⤴ SQL funktsiooni kasutava lause täitmine oli kiirem kui tabeli ühendamist sisaldava lause täitmine.

Päring, mis kasutab tabelite välisühendamist: **6982 ms**

Päring, mis kasutab SQL funktsiooni: **4921 ms**

Päring, mis kasutab PL/pgSQL funktsiooni: **5573 ms**

Selle ülesande lahendamiseks saab lisainfot slaidikomplektidest:

- ⤴ **Domeenid_vaated_synonyymid_IDU0230_2018.ppt**
 - ⤴ **SQL_DML_alamkeelee_voimalusi_IDU0230_2018.ppt**
 - ⤴ **PostgreSQL_operaatorid_SQL_funktsioonid_IDU0230_2018.ppt**
- , mille leiab <http://193.40.244.90/346>