# Ülesanne 6

Selle ülesande lahendamiseks saab lisainfot slaidikomplektist:

- **Domeenid_vaated_synonyymid_IDU0230_2018.ppt, mille leiab http://193.40.244.90/346** kataloogist:
  *Andmebaasid 2/Andmebaasisüsteemide juhendid ja koodinäited/Slaidid andmebaasiobjektide kohta*

Seda ülesannet saavad lahendada need, kes teevad andmebaasi PostgreSQLis, kuid ei saa teha need, kes teevad andmebaasi Oracles. PostgreSQLis on võimalik luua SQL domeene, kuid Oracles ei ole. Oracles projekti tegijad peaksid siiski ülesande ning sellega seotud materjalidega tutvuma.

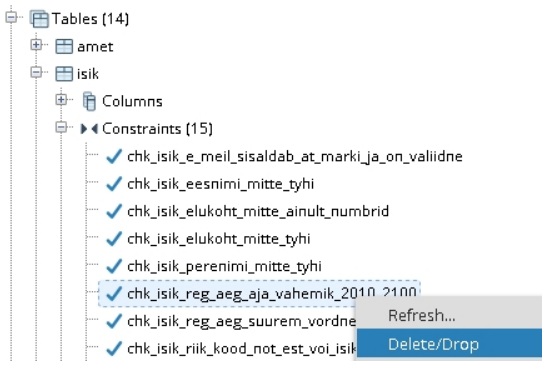
Ülesandeks on andmebaasi refaktoreerimine (http://databaserefactoring.com/), mille sisuks on samasisuliste veergude puhul domeeni kasutuselevõtt nende veergude omaduste kirjeldamiseks. Hindamismudeli kohaselt tuleb projektis luua **vähemalt üks domeen. Kõiki loodud domeene tuleb andmebaasis kasutada ning seda peab tegema järjekindlalt, st peab kasutama kõikjal kus on samasuguse omadusega veerud.**

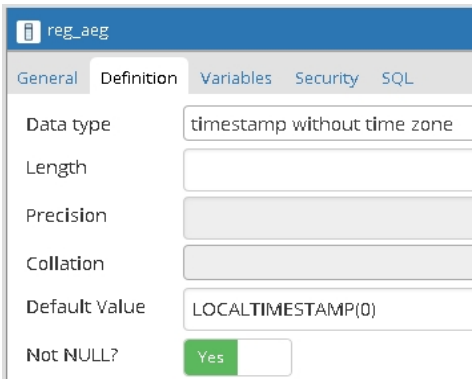
**Refaktoreerimine ei tohi tehja kahju. Andmebaasi refaktoreerimise tulemusena ei tohi minna kaotsi andmebaasis olemasolevad andmed! Konkreetsel juhtumil ei tohi minna kaotsi ülesande 5 tulemusena andmebaasi lisatud andmed.**

**Mis peab olema ülesande tulemusena tehtud?**

- Domeenide loomise laused PostgreSQLis – CREATE DOMAIN
  https://www.postgresql.org/docs/11/static/sql-createdomain.html

- Muudetud andmebaas. Täpsemalt tuleb teha muudatused, mille põhimõtet on selgitatud slaidikomplekti "Domeenid, vaated, sünonüümid" andmebaasi refaktoreerimise kohta käival slaidil ning selle kommentaaride lehel. Veergudega seotud CHECK kitsenduste kustutamisel pidage silmas, et kustutada on vaja ainult need kitsendused, mis on defineeritud domeenide kaudu.
  - Nende muudatuste tegemiseks võite kirjutada SQL lauseid (nagu slaidi märkmelehel olevas näites). PostgreSQL – ALTER TABLE
    https://www.postgresql.org/docs/11/static/sql-altertable.html
  - Kahjuks ei saa muudatuste tegemiseks kasutada *ainult* pgAdmin III ja pgAdmin 4 ver 3 (või varasem) graafilist kasutajaliidest. Seal *ei ole* võimalik muuta olemasoleva tabeli veeru spetsifikatsiooni nii, et veerg oleks defineeritud domeeni põhjal. Samas saab nende graafiliste kasutajaliideste kaudu kustutada veeruga seotud CHECK ja NOT NULL kitsendusi ning vaikimisi väärtuseid. Seega võib kasutada strateegiat, et veeru ja domeeni sidumiseks kirjutate ALTER TABLE lause, kuid ülejäänu teete graafilise kasutajaliidese kaudu.




- Muudetud CREATE TABLE laused (ülesande 4 tulemus), mis kajastavad muudatusi andmebaasis. CREATE TABLE lausetesse tuleb lisada viited kasutuselevõetud domeenidele ning neist tuleb eemaldada kitsendused ja vaikimisi väärtused, mis määratakse veergudele domeenidega. Eesmärgiks on saada CREATE TABLE laused, mille käivitamisel luuakse domeene kasutavad tabelid.

```
CREATE TABLE Vastuvott (
vastuvott_id SERIAL,
ruum_id INTEGER NOT NULL,
tootaja_id INTEGER NOT NULL,
kuupaev d_kuupaev_kohustuslik,
kommentaar d_kirjeldus,
CONSTRAINT Vastuvott_id_on_primaarvoti PRIMARY KEY (vastuvott_id),
CONSTRAINT Vastuvott_toimub_ruumis FOREIGN KEY (ruum_id) REFERENCES Ruum
(ruum_id),
CONSTRAINT Tootaja_votab_vastu FOREIGN KEY (tootaja_id) REFERENCES Tootaja
(tootaja_id)
) WITH (fillfactor=90);
```

- Laused loodud domeenide kustutamiseks (need on tuleb samuti projekti dokumendis esitada), kuid neid pole vaja käima panna.

PostgreSQL – DROP DOMAIN
https://www.postgresql.org/docs/11/static/sql-dropdomain.html

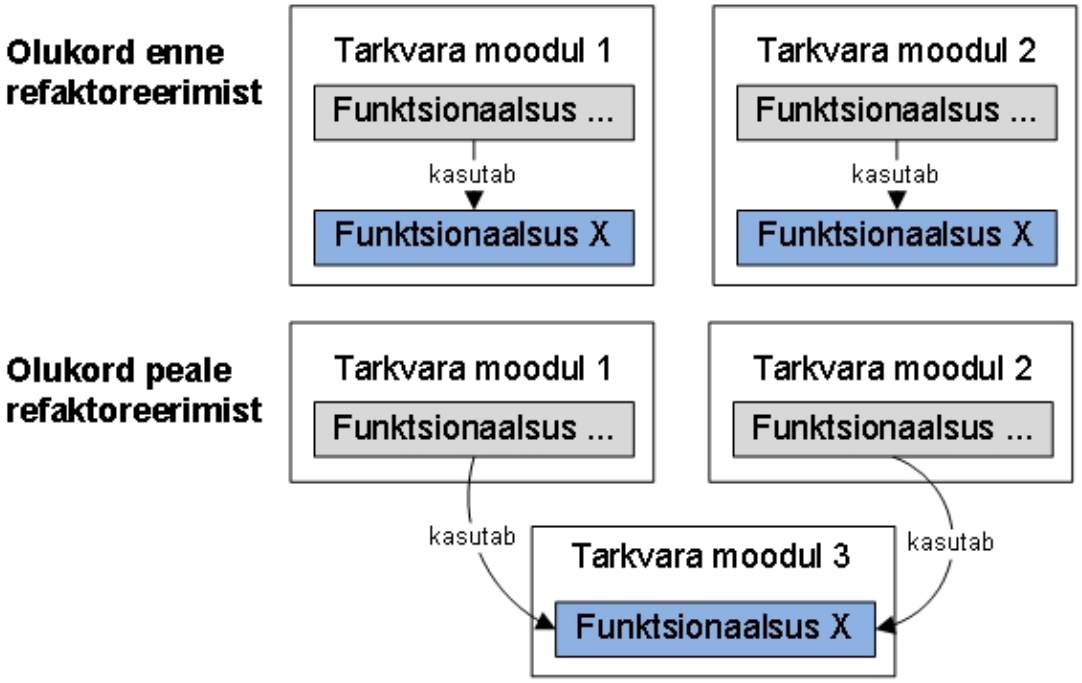
- Kuna Enterprise Architect ja Rational Rose CASE vahendid ei võimalda domeene kirjeldada ning nende loomise koodi genereerida, siis andmebaasi disaini mudelites ei ole vaja muudatusi teha.

- Andmebaasis on alles sinna ülesande 5 tulemusena lisatud andmed. Nende säilitamiseks on tehtud *võimalikult vähe* lisatööd. Kui järgite eeltoodud juhendit (andmebaasis olevate tabelite muutmine), siis andmed säilivad. Alternatiivne, aga **palju halvem** variant (sest protsess võtab rohkem aega ja kui parandada on vaja kasutuses olevat andmebaasi, siis sunnib seda kasutavaid süsteeme mõneks ajaks väljalülitama) on järgmine.
  - Tabelites juba olemasolevate lausete eksportimine INSERT lausetena. Vaadake selle kohta kataloogis *Tarkvara saamine ja kasutamine/PostgreSQL videod (PostgreSQL 9.3 baasil)* olevaid videoid

        - PostgreSQL andmebaasi loogilise varukoopia tegemine kasutades pgAdmin programmi
        - PostgreSQL andmebaasi loogilise varukoopia tegemine kasutades shelli promptist pg_dump programmi
    - Kõigi eelnevate ülesannete töö tulemustest tervikliku andmebaasi loomise stkripti kokkupanemine. Skripti alguses on laused varem loodud andmebaasiobjektide kustutamiseks. Skriptis on ka väliste tabelite loomise laused ning kõige lõpus INSERT laused.
    - Andmebaasi skripti käivitamine.

Domeenide kasutuselevõtu eesmärgid.

- Ühtlustada samasisuliste andmetega veergude omadusi.
- Luua koht, kus muudatuse tegemine tingib muudatused kõigi nende veergude omadustes. Tõmmake paralleele järgnevaga.




Domeeni probleeme.

- Peale domeeni loomist ei ole võimalik muuta selle baastüüpi ega maksimaalset väljapikkust (PostgreSQL ALTER DOMAIN lause võimaldab rohkem kui SQL standard ette näeb, kuid neid tegevusi ikkagi ei luba). https://www.postgresql.org/docs/11/static/sql-alterdomain.html
    - SQL standardi kohaselt tingib DROP DOMAIN d CASCADE domeeni *d* kustutamise. Kui on mõni veerg, mis on *d* põhjal defineeritud, siis veerg jääb alles. Veeru tüüp, vaikimisi väärtus, NOT NULL ja CHECK kitsendused määratakse *d* põhjal ja seotakse otse veeruga. PostgreSQLis tingib DROP DOMAIN d CASCADE kõikide *d* põhjal defineeritud veergude tabelist kustutamise. Seega domeeni baastüübi muutmise stsenaariumis ei saa DROP DOMAIN ... CASCADE kasutada.
- SQL domeen ei ole tüüp ning seega ei toeta tugevat tüüpimist (vt teema 2 materjale).

**Disaini soovitus**: Kaaluge domeeni loomist, kui andmebaasi tabelites on *vähemalt kaks* veergu, mille omaduste kirjeldamiseks seda domeeni võiks kasutada ning domeeniga on seotud üks või rohkem CHECK kitsendust ja/või märkide võrdlusreeglistik (*collation*) ja/või vaikimisi väärtus. Üldine põhimõte: mida rohkem mingit tehist taaskasutatakse, seda rohkem "teenitakse tagasi" selle loomisele kulutatud vahendeid.

Näited, millised võiksid olla refaktoreerimise kandidaadid.

- Klassifikaatori nimetus (kohustuslik tekstistring, mis ei tohi olla tühi ega koosneda ainult tühikutest).
- Kommentaar/kirjeldus (mittekohustuslik pikk tekstistring, mis ei tohi olla tühi ega koosneda ainult tühikutest).
- Loomise aeg (kohustuslik kuupäeva või ajatempli tüüpi väärtus, mille vaikimisi väärtus leitakse funktsiooniga).
- Rahasumma (kohustuslik kümnendmurd, täpsusega kaks või rohkem kohta peale koma, vaikimisi väärtusega 0 ning kitsendusega, et väärtus ei tohi olla väiksem kui 0).
- Kogus (kogustuslik väike täisarv, vaikimisi väärtusega 0 ning kitsendusega, et väärtus ei tohi olla väiksem kui 0).

Eelnevalt väljatoodud kitsendused ja vaikimisi väärtused on *näited* – milliseid on vaja kasutada Teie projektis, tulenevad selle projekti nõuetest.

**Mis on järgmise lahenduse probleem?**

```
CREATE DOMAIN d_loomise_aeg
TIMESTAMP NOT NULL DEFAULT LOCALTIMESTAMP(0)
CONSTRAINT chk_d_loomise_aeg CHECK (VALUE BETWEEN '2010-
01-01 00:00:00' AND '2099-01-01 23:59:59');
```

```
CREATE TABLE Kaup
(...
loomise_aeg d_loomise_aeg NOT NULL DEFAULT
LOCALTIMESTAMP(0),
...
CONSTRAINT chk_Kaup_loomise_aeg CHECK (loomise_aeg BETWEEN
'2010-01-01 00:00:00' AND '2099-01-01 23:59:59'));
```

Kitsendusi (NOT NULL ja CHECK) ning vaikimisi väärtust (DEFAULT) korratakse antud juhul koodis kahes kohas. Täpselt nagu liiasusega andmete muutmisega – kui on vaja teha muudatus süsteemi käitumises, siis peab seda tegema mitmes kohas. See nõuab rohkem tööd ning kui muudatus ühes kohas tehakse ning teises kohas mitte, siis tekib süsteemi käitumisse vastuolu. Õige oleks lähtuda disainiprintsiibist "Ainult üks kord" (Once and Only Once, http://wiki.c2.com/?OnceAndOnlyOnce) ning luua tabel *Kaup* järgnevalt.

```
CREATE TABLE Kaup
(...
loomise_aeg d_loomise_aeg,
...);
```

Ka Teie peate oma projektis sellisest dubleerimisest hoiduma.

**Mis on järgmise lahenduse probleem?**

```
CREATE DOMAIN d_viide_klassifikaatorile
SMALLINT NOT NULL DEFAULT 1;
```

```
CREATE TABLE Osapool
(...
osapoole_seisundi_liik_kood d_viide_klassifikaatorile,
...);
```

```
CREATE TABLE Tootaja
(...
tootaja_seisundi_liik_kood d_viide_klassifikaatorile,
...);
```

Nimetatud veerud kuuluvad ühtlasi välisvõtmetesse, mis viitavad *erinevatele* seisundiklassifikaatorite tabelitele. Nende seisundiklassifikaatorite abil registreeritakse, milline on iga konkreetse töötaja ja osapoole hetkeseisund tema elutsükli kohaselt. Elutsüklid kirjeldavad protsesse. Protsessid võivad ajas muutuda (evolutsioneeruda), sest näiteks organisatsioon ja tänu sellele ka tema infosüsteem peab tänu muutunud ärikeskkonnale või seadusandlusele, hakkama teistmoodi käituma. *Mida ma peaksin tegema, kui töötajate elutsükli mudeli kohaselt on edaspidi töötaja algseisundiks seisund koodiga 2?*

1. Kustutama vaikimisi väärtuse domeenist *d_viide_klassifikaatorile*
2. Määrama vaikimisi väärtuse tabelis *Osapool* veerule *osapoole_seisundi_liik*
3. Määrama vaikimisi väärtuse tabelis *Tootaja* veerule *tootaja_seisundi_liik*

See on kolm sammu, võrreldуna ühe sammuga – muuda vaikimisi väärtust tabelis *Tootaja* veerul *tootaja_seisundi_liik*, mida peaksin tegema, kui selliselt domeeni kaudu vaikimisi väärtust ei määra.

Laiemalt võttes eksib domeen *d_viide_klassifikaatorile* otstarbe lahususe (*separation of concern*) disainipõhimõtte (https://en.wikipedia.org/wiki/Separation_of_concerns) vastu, sest ühte tehisesse (antud juhul domeeni) on koondatud vastutus määrata ära erinevat tüüpi põhiobjektide algseisund. Antud juhul ei anna domeeni loomine midagi juurde, tekitab vaid edasiseks andmebaasi halduseks probleeme. Parem oleks see loomata jätta ning siduda vaikimisi väärtus otse veeruga.

**Miks on vastuvõtuaegade näiteprojektis nii palju domeenide kirjeldusi ja loodud andmebaasis nii palju domeene?**

Ühes vanemas näiteprojekti versioonis olid kaks erinevat tabelit – *Yliopilane* ja *Tootaja*, kus olid veerud *eesnimi*, *perenimi* jne. Nende jaoks said ka loodud domeenid. Andmebaasi evolutsioneerudes tekkis ühine tabel *Isik,* kuna on selge, et ülikooli töötaja võib olla ka samal ajal üliõpilane (magistrant või doktorant). Kuna ma ei näinud põhjust domeenidega tehtud töö äraviskamiseks (halvemaks nad ka midagi ei tee), siis säilitasin need domeenid ka andmebaasi uue versiooni korral. Lisaks olid mõned domeenid defineeritud ka teise

näiteprojekti (õpingukavade arvestus) jaoks ning kasutan neid ka selles projektis.

Kui hakkaksin seda projekti uuesti tühjalt kohalt tegema, siis ei looks ma kõiki selliseid domeene ja ka Teie ei pea seda tegema (aga võite teha kui tahate – miinuseid selle eest ka ei saa). Praegu seda projekti uuest tehes looksin vaid domeenid *d_nimetus* ja *d_kirjeldus*, sest kasutan neid mitme veeru korral.

Samas on võimalik ühe projekti käigus defineeritud domeene kasutada teistes projektides, teiste andmebaaside kirjeldamisel ning selle kaudu domeenide kirjeldamise vaev "tagasi teenida".

**Domeeni loomine teise domeeni põhjal**

See on PostgreSQLis võimalik kuigi nt SQL:2011 standardi versioon seda ette ei näe.

```
CREATE DOMAIN d_tekst TEXT;

CREATE DOMAIN d_amet_kood d_tekst NOT NULL
CONSTRAINT chk_amet_kood_pikkus CHECK (length(VALUE)<4);

CREATE TABLE Amet (amet_kood d_amet_kood,
CONSTRAINT pk_amet PRIMARY KEY (amet_kood));

INSERT INTO Amet (amet_kood) VALUES ('123');
--Lisamine õnnestub

INSERT INTO Amet (amet_kood) VALUES ('1234');
/*ERROR:  value for domain d_amet_kood violates check
constraint "chk_amet_kood_pikkus"*/
```

Õppejõu soovitus on seda võimalust mitte kasutada, sest see tekitab liigset keerukust ja lisab sõltuvusi, mida tuleb hakata jälgima. Kui muudan domeeni *d_tekst* omadusi, siis see võib mõjutada mitut erinevat domeeni ja selle kaudu senisest rohkem veerge. Mida rohkem on sõltuvusi, seda lihtsam on teha vigu muudatustest tuleneva mõju hindamisel. Samuti võivad sisse lipsata vead, mille tulemusena erinevatel üksteisega seotud domeenidel on üksteisega vastuolus olevad kitsendused.

```
ALTER DOMAIN d_tekst ADD CONSTRAINT chk_tekst_min_pikkus
CHECK (length(VALUE)>=4);
```

Nüüd ei saa uusi ameteid lisada, või olemasolevate koode muuta, sest kood peab olema korraga vähem kui neli märki pikk ja neli või rohkem märki pikk.

**Tüüpidest ja domeenidest**

Kui SQL võimaldaks defineerida uusi tüüpe niisama lihtsalt kui praegu domeene, siis ma looksin iga näiteprojektis kirjeldatud domeeni *asemel* tüübi. See tagaks, et kui üritan näiteks päringut:

```
SELECT *
FROM Isik INNER JOIN Amet ON Isik.eesnimi=Amet.nimetus;
```

, siis sellise lause täitmine ebaõnnestuks, sest puudub seda tüüpi väärtuste võrdlemiseks mõeldud operaator. Enne kui hakata sellist operaatorit looma on võimalus järgi mõelda, kas sellist võrdlust on ikka vaja teha. Ilmselt jõuaksin peale järelemõtlemist järeldusele, et ei ole ning seega pole vaja ka operaatorit luua. Eelnev oleks *tugeva tüüpimise* näide. Tugev tüüpimine tähendab, et

- igal väärtusel, avaldisel ja muutujal on tüüp,
- iga kord, kui proovime teha väärtuse, avaldise või tüübiga mõne operatsiooni, kontrollib süsteem, kas osalised on operatsiooni läbiviimiseks sobivat tüüpi (st, kas leidub operatsiooni läbiviimiseks sobivat tüüpi parameetritega operaator).

Kui veerud *eesnimi* ja *nimetus* on defineeritud SQL domeeni abil, siis see päring veateadet ei anna ning täidetakse, sest mõlema domeeni baastüüp on ilmselt VARCHAR ning päringu täitmiseks võrreldakse VARCHAR tüüpi väärtuseid.

**Domeenidega seotud kitsendused**

Kuidas kontrollida, kas domeeniga seotud CHECK kitsendus on õigesti realiseeritud?

**Vastus**: Tuleks käivitada SELECT lauseid, kus literaali abil esitatud väärtust üritatakse teisendada domeenile vastavaks. Tuleks katsetada nii väärtustega, mille puhul see teisendus peaks õnnestuma kui ka väärtusega, mille puhul mitte.

Näide: Proovige andmebaasis *vastuvotuajad* järgmiseid lauseid. **::d_e_mail** osa lausetes algatab soovitud teisenduse.

```
SELECT 'Erki.Eessaar_at_ttu.ee'::d_e_mail;
/*Täitmine ebaõnnestub.
ERROR:  value for domain d_e_mail violates check
constraint "e_mail_peab_sisaldama_tapselt_yhte_at_marki"*/


SELECT 'Erki.Eessaar@ttu.ee'::d_e_mail;
/*Täitmine õnnestub, st e-meili aadress vastab
defineeritud reeglitele - sisaldab täpselt ühte @ märki*/

SELECT 'Erki.Eessaar@ttu@ee'::d_e_mail;
/*Täitmine ebaõnnestub.
ERROR:  value for domain d_e_mail violates check
constraint "e_mail_peab_sisaldama_tapselt_yhte_at_marki"
*/
```

**Domeenide identifikaatorid e nimed.**

- Ärge andke domeenie sama nime kui andmebaasis oleval andmetüübil. Sellega tekitate suurt segadust.

```
CREATE DOMAIN text AS VARCHAR(3);

CREATE TABLE Proov(
veerg1 text, --veerg on tüüpi text
veerg2 public.text /*veerg on defineeritud domeeni text
põhjal, mis on skeemis public*/);

DROP DOMAIN text;
/*ERROR:  "text" is not a domain*/

DROP DOMAIN public.text;
/*Kustutamisel tuleb kindlasti osutada skeemile.*/
```

- PostgreSQLis luuakse tabeli (nii baastabeli, vaate kui hetktõmmise) loomisel samas skeemis automaatselt andmetüüp (liittüüp), mille nimi on samasugune kui tabeli nimi. Ühes skeemis ei saa olla sama nimega domeen ja andmetüüp. Seega ei saa skeemis ka olla samasuguse nimega domeeni ja tabelit.
- Andke domeenidele sisukad nimed.
- Andke domeendes sisalduvatele CHECK kitsendusele sisukad nimed ning kasutage üldiselt samasugust nimetamise stiili nagu otse tabelitega seotud CHECK kitsenduste korral.
  - Kuna iga domeeni võib kasutada mitme tabeli veergude kirjeldamisel ning nende veergude hulk võib peale domeeni loomist muutuda, siis ei ole mõistlik domeeni CHECK kitsenduse nimes kasutada tabelite ja veergude nimesid.
- Soovitan kasutada **[S/s]nake_case**.
- Nimed peavad vastama PostgreSQL reeglitele.
  https://www.postgresql.org/docs/11/static/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS