Ülesanne 3

Eesmärgiks on iseseisva töö projekti andmebaasi füüsilise disaini mudelis ära kirjeldada kõik CHECK kitsendused ja välisvõtmetele loodavad indeksid. Vajadusel tuleb kooskõla säilitamiseks täiendada kontseptuaalset andmemudelit. Tööd tuleb teha oma projekti kaaslastega koos rühmatööna.

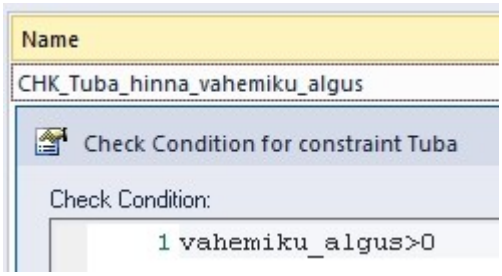
Alustuseks vaadake videot "CHECK kitsenduste kirjeldamine andmebaasi diagrammis" (kataloog *Tarkvara saamine ja kasutamine*). Vaadake CHECK kitsenduste stiilinäiteid kataloogis *Töö harjutustunnis (samm-sammuline juhend)*.

Kitsenduste (sh CHECK kitsenduste) andmebaasis jõustamine on üks **kaitsva andmebaaside disaini** (*defensive database design*) praktikatest: https://www.red-gate.com/simple-talk/sql/database-administration/developing-low-maintenance-databases/

"Pole olemas rumalaid kasutajaid, on ainult rumalad programmid" (mis lasevad kasutajatel teha rumalusi) (http://blog.williams-helde.com/there-is-no-such-thing-as-dumb-user-there-are-only-dumb-products). Seda võib nentida seoses turvalisuse tagamisega, aga tegelikult ka seoses andmete reeglitele vastavuse kontrollimisega (mille saavutamiseks on CHECKid üks otstarbekas viis). Andmebaasis deklareeritud kitsendused (sh CHECKid) on head, sest ei lase kasutajatel rumalusi teha ja ennast selle kaudu kahjustada.

CHECK kitsenduste kirjeldamisest CASE vahendis

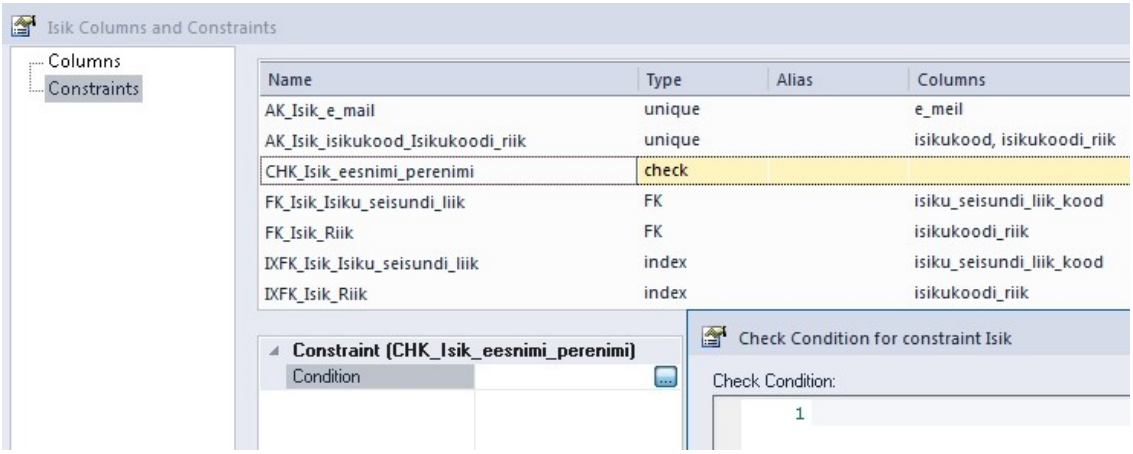
- Kitsenduste avaldiste kirjeldamisel EA või RR vahendis pöörake tähelepanu sellele, et avaldistes kasutatavad veergude nimed ja tabeli kirjeldustes kasutatavad veergude nimed langeksid kokku. Paraku need CASE vahendid seda ise ei kontrolli. Tõehetk saabub siis, kui hakkate mudelist genereeritud koodi käivitama – siis võib selguda, et avaldis on ebakorrektne ning tabeli loomise lauset ei täideta. **EA ja RR ei kontrolli CHECK kitsenduste avaldisi.** Näide: avaldis viitab veerule *vahemiku_algus*, kuid tabelis on veerg *hinna_vahemiku_algus*. Kui muudate veeru nime, siis tuleb käsitsi kõik sellele veerule viitavad CHECK kitsendused üle vaadata ning avaldisi muuta.



- EAs ei ole CHECK kitsenduste kirjeldamisel vaja määrata *Assigned Columns*. Kui määrate, siis see pole viga, kuid see ei mõjuta koodi genereerimist.

- CHECK kitsendusse kirjutatakse samasugune tingimus (loogikaavaldis, predikaat), nagu SQL andmekäitluskeele lausete WHERE klauslisse.
  - Avaldises peab olema veeru nimi ja see peab langema kokku tabeli kirjelduses kasutatud veeru nimega.
  - Avaldises peab olema korrektne pöördumine mingi võrdluse operaatori poole.
  - **Valesti** kirjutatud avaldiste *näited*.
    - `0`
      - puudub veeru nimi ja võrdluse operaator
    - `(eesnimi IS NOT NULL) | (perenimi IS NOT NULL)`
      - | asemel on SQLis operaatori nimi OR.
    - `e_mail ALIKE '%@%'`
      - PostgreSQLis ja Oracles pole ALIKE operaatorit (on LIKE).
    - `synni_kp Between #1.01.1900# And #31.12.2100#`
      - PostgreSQLis ja Oracles on kuupäevastringid ülakomade (apostroofide) vahel.
      - PostgreSQLis peab kuupäevastringi esitama kujul 'YYYY-MM-DD' (võib ka kasutada *to_date* funktsiooni).
      - Oracles peab kuupäeva esitama kasutades *to_date* funktsiooni.
    - `'2010-01-01' <=synni_kp<= '2099-12-31'`
      - Siis peab olema kaks alamtingimust, mis on AND abil kokku pandud või üks tingimus, mis kasutab BETWEENi.
    - `[synni_kp]>='2010-01-01'`
      - Veergude nimede kirjutamine nurksulgudesse on lubatud MS Accessis, kuid pole lubatud PostgreSQLis ja Oracles.

Kuidas lisada CHECK kitsenduse kirjeldust EA12 vahendis?



- Avate tabeli kirjelduse.
- Valite *Table Detail.*
- Klõpsate nupul *Constraints.*
- Lisate nimekirja lõppu uue kitsenduse, määrates nime ja tüübi (*check*). Pange tähele, et peale nime sisestamist sorteeritakse kitsenduste nimekiri automaatselt ümber, et nimed oleksid tähestikulises järjekorras.
- Lisate kitsenduse tingimuse (*Condition*) (klõpsake välja taga nupul kus on kolm punkti).

Millele CHECK kitsenduste kirjeldamisel tähelepanu pöörata?

- Kõik kitsendused tulenevad kontseptuaalses andmemudelis olemitüüpidele, atribuutidele ja seosetüüpidele kirjeldatud kitsendustest.

Hoone	nimi	Hoonet unikaalselt identifitseeriv nimi, mida kasutatakse hoonele viitamiseks ka väljapool andmebaasi.	IT-maja õppehoone
		{Hoone unikaalne identifikaator. Registreerimine on kohustuslik. Nimi ei tohi olla tühi string või ainult tühikutest koosnev string.}	

Selle põhjal loon **UNIQUE**, **NOT NULL** ja **CHECK** kitsendused.

- Mõnede kitsenduste puhul saab MS Accessis kirjutatud valideerimisreegli üsna üks-ühele serveri andmebaasi üle tõsta.
  - Mõnede kitsenduste puhul saab MS Accessis kirjutatud valideerimisreegli serveri andmebaasi üle tõsta, lisades CHECK kitsenduse loogikaavaldisse veeru nime (nimed).
    - MS Access: `>0`
    - PostgreSQL või Oracle: `hind>0`
  - MS Accessis loodud mitmeosalised tabelitaseme valideerimisreeglid tuleb serveri andmebaasis asendada mitme CHECK kitsendusega, millest igaühel on kitsam ülesanne.
  - Kui on vaja kontrollida teksti vastavust mustritele, siis pakuvad PostgreSQL ja Oracle võrreldes MS Accessiga suuremaid võimalusi, sest seal saab kasutada regulaaravaldisi.
- Kui tahate andmebaasi disaini/realiseerimise käigus mõne uue kitsenduse kirjeldada, siis tuleb see ka lisada kontseptuaalsesse andmemudelisse. Teistpidi – kui mõni kontseptuaalses andmemudelis esitatud kitsendus osutub Teie jaoks liiga keeruliseks, siis on võimalik see kontseptuaalsest andmemudelist eemaldada või seda lihtsustada.
  - Näiteks võib tekkida mõte, et juhul kui projekti ärireeglid näevad ette, et kõik praegu ja tulevikus registreeritavad isikud on kas Eesti kodanikud või Eestis viibivad välismaalased, siis saaks lähtuda veergudele *eesnimi* ja *perenimi* kitsenduste määramisel Eesti Vabariigi nimeseaduse paragrahvidest 6 ja 7 (https://www.riigiteataja.ee/akt/106032015032?leiaKehtiv). Samas on nendes paragrahvides ka lõigud, mis nendivad, et nendesse reeglitesse võib teha erandi kui "isikul on oma kodakondsuse, peresuhete, rahvuskuuluvuse või usu tõttu isiklik seos muukeelse nimetraditsiooniga ja taotletav nimi on sellele vastav." Tabeliga seotud kitsendused kehtivad kõigile selle tabeli ridadele. Erandite lubamine tähendab, et selliseid kontrolle ei ole õige realiseerida andmebaasi tasemel olevate kitsendustena. Samas võib need realiseerida isikute registreerimise rakenduses, kus need annavad andmete registreerijale hoiatusi, kuid ei lükka andmeid tagasi.
  - Leiate materjalidest regulaaravaldise, mis kontrollib isikukoodi vastavust Eesti isikukoodi reeglitele. Kui Teie projekt lubab registreerida isikukoode erinevatest riikidest, siis ei saa Te seda kontrolli üks-ühele kasutada. Aga, kui soovite **(pole kohustuslik)**, saate andmebaasis jõustada reegli *"Kui isikukoodi riik on Eesti, siis isikukood peab vastama mustrile ..."*. Formaalsemalt kirja panduna

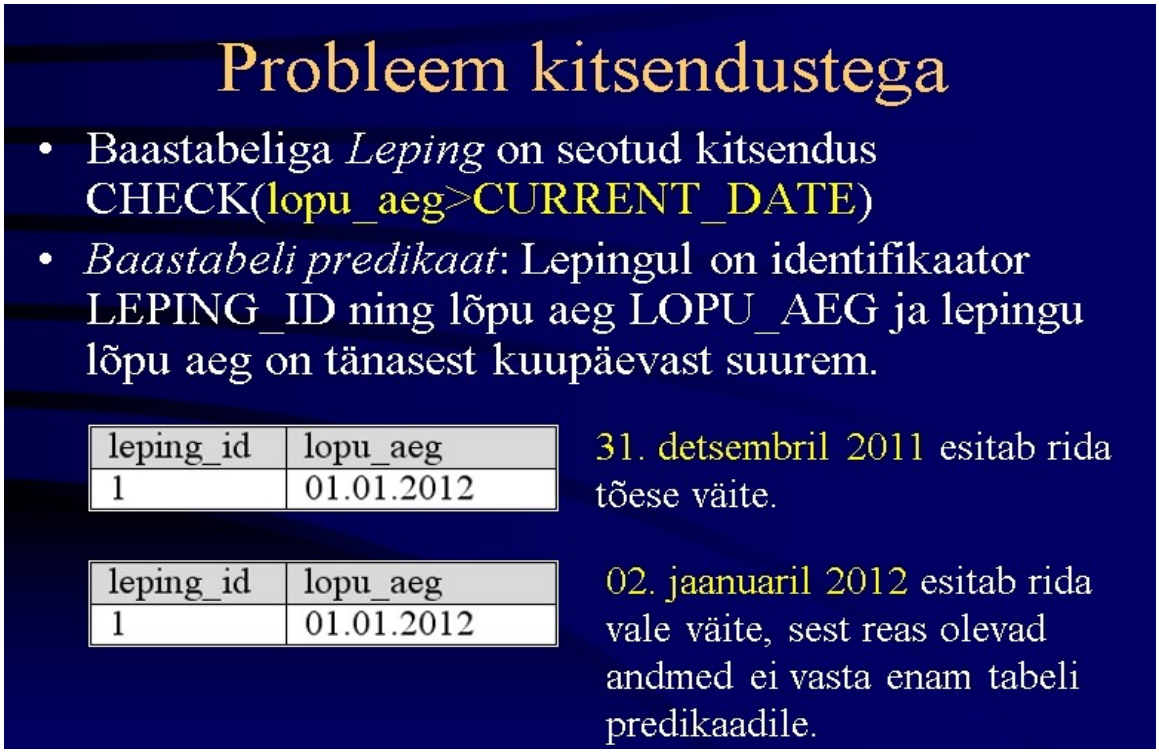
on tegemist implikatsiooni reegliga, mille üldkuju on selline **P=>Q**. See tähendab, et kui kehtib tingimus P, siis peab kehtima ka tingimus Q. Sellise reegli CHECK kitsendusena jõustamiseks, saate kasutada avaldisega P=>Q loogiliselt samaväärset avaldist **NOT(P) OR Q**.

- PostgreSQLis oleks kitsenduse loogikaavaldis selline:
  - `NOT (riik_kood='EST') OR isikukood~'^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-3]{1}[[:digit:]]{5})$'`
- Oracles oleks kitsenduse loogikaavaldis selline:
  - `NOT (riik_kood='EST') OR REGEXP_LIKE(isikukood, '^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-3]{1}[[:digit:]]{5})$')`

Kopeerimise järel eemaldage reavahetused.

Kuna põhimõtteliselt võiks iga riigi jaoks lisada sellise eraldi kitsenduse, siis peaks kitsenduse nimi, viitama konkreetsele riigile (Eesti, EST)

- Hindamismudeli kohaselt kaotatakse punkte järgmise vea eest.
  - Vähemalt kaks CHECK kitsendust, mis peaksid loogiliselt andmebaasis olema, kuid mida ei kirjelda kontseptuaalne andmemudel ja neid pole ka loodud andmebaasis (näited: **kogust näitavate arvude puhul lubatud väärtuste vahemik; kuupäevade puhul lubatud kuupäevade vahemik; kuupäevade paaride puhul näidatud, milline peab olema nende kuupäevade järjestus**) (tüüpvead 29, 35)
    - Teiste sõnadega, kui Teie andmebaasis on *näiteks* veerg *hind,* kuhu saab lisada negatiivseid väärtuseid, siis miinuspunktide eest ei saa kõrvale põigata väitega, et "Kontseptuaalses andmemudelis polnud seda piiravat kitsendust kirja pandud". See kitsendus peab olema seal ja peab olema Teie andmebaasis.
  - See probleem puudutab eeskätt eeldusaines arvestamata projekte. Siiski peaksite kõik veenduma, et Teie kontseptuaalses andmemudelis ja andmebaasi disaini mudelis on vajalikud kitsendused olemas. Kui ei ole, siis lisage. Näiteks, kui lisasite tabelitesse veerge, siis uuendage kontseptuaalset andmemudelit ja mõelge kitsendustele.
  - Hea näite kahe silma vahele jäänud kitsendusest toob Amazoni boss ja maailma üks rikkamaid inimesi Jeff Bezos. Amazoni algusaastatel avastas ta oma süsteemist vea, mis lasi kliendil tellida negatiivse arvu raamatuid, mille järel kanti raha kliendi krediitkaardile. Hr Bezos viskab ühel videol nalja, et ilmselt tuli siis jääda ootama, kuni klient neile raamatud saadab. https://www.businessinsider.com/when-amazon-launched-a-bug-allowed-users-to-get-paid-by-the-company-2011-10
- PostgreSQL ja Oracle ei võimalda kasutada CHECK kitsendustes alampäringuid.
- PostgreSQL võimaldab kasutada CHECK kitsendustes kasutaja-defineeritud funktsioone, Oracle ei võimalda.
- PostgreSQL võimaldab kasutada CHECK kitsendustes mittedeterministlikke funktsioone (nt CURRENT_DATE), Oracle ei võimalda. Mittedeterministlik funktsioon annab *samade argumentidega* erinevatel ajahetkedel käivitades erineva tulemuse.
  - Kui plaanite kasutada CHECK kitsenduses mõnda mittedeterministlikku funktsiooni, siis ärge looge kitsendusi, mis tekitavad järgneval slaidil tutvustatud probleemi.



- Kui kontseptuaalses andmemudelis on mõni selline kitsendus, millele vastavalt ei looda andmebaasis deklaratiivset kitsendust, siis tuleb see kitsendus jõustada kasutades protseduurset viisi (näiteks trigerite abil; PostgreSQLis saab kasutada ka reegleid). Jätke sellised kitsendused meelde, et saaksite neid hiljem trigerite/reeglite loomise juures kasutada.
  - Eelneval slaidil illustreeritud kitsendus (sisestamise hetkel peab lõpu aeg olema tulevikus) tuleb samuti jõustada kas trigeriga, andmebaasi avalikus liideses (protseduuris/funktsioonis) või rakenduses.
- Ärge kasutage kitsenduste loogikaavaldistes kahekordset eitust – see muudab koodi halvemini mõistetavaks. Näide:
  - **Halb** (ei tohi olla nii, et e-meili aadress ei sisalda @ märki):
    - `CHECK(NOT (e_meil NOT LIKE '%@%'))`
  - **Parem** (peab olema nii, et e-meili aadress sisaldab @ märki):
    - `CHECK(e_meil LIKE '%@%')`
- Ärge defineerige ühte ja sama kitsendust mitu korda – see on koodi (käitumise) dubleerimine ja tekitab asjatuid haldusprobleeme. Duplikaatide vältimisel pidage meeles, et sageli saab ühe ja sama avaldise kirja panna mitmel erineval viisi, st ei piisa sellest, et väldite otseseid koodi korduseid, vaid korduste vältimisel tuleb vaadata ka sisu.
  - Näide: kitsenduste loogikaavaldiste komplektid, mis jõustavad sama reegli, et nimetus ei tohi olla tühi string ega tühikutest koosnev string.
    - Komplekt 1:
      - `nimetus!~'^[[:space:]]*$'`
    - Komplekt 2: (Ei sobi Oracle jaoks, sest Oracle asendab "=> NULL)
      - `trim(nimetus)<>''`
    - Komplekt 3: (Ei sobi Oracle jaoks, sest Oracle asendab "=> NULL)
      - `nimetus<>''`
      - `nimetus!~'^[[:space:]]+$'`

- Näide:
  - Kui veerus peab olema alati väärtus, siis selle tagamiseks tuleb veerule defineerida NOT NULL kitsendus. Eraldi CHECK kitsendust selleks vaja ei ole.
    - **Halb**:
      - `... isikukood VARCHAR(50) NOT NULL, ... CHECK (isikukood IS NOT NULL)`
      - `... isikukood VARCHAR(50), ... CHECK (isikukood IS NOT NULL)`
        - Pange kood sinna, kus teised lugejad seda kõige tõenäolisemalt eeldavad olevat.
    - **Parem**:
      - ... isikukood VARCHAR(50) NOT NULL, ...
- Välisvõtme veergudes pole vaja korrata viidatud kandidaatvõtme veergudele jõustatud kitsendusi. Välisvõti tähendab, et selle väärtus peab olema üks kandidaatvõtme väärtustest.

Näide:
Riik (riik_kood, nimetus)
Primaarvõti (riik_kood)
Alternatiivvõti (nimetus)

Isik(isik_id, isikukood, riik_kood, eesnimi)
Primaarvõti (isik_id)
Alternatiivvõti (isikukood, riik_kood)
Välisvõti (riik_kood) viitab Riik (riik_kood)

Riigi koodile on mõtet CHECK kitsendus (nt peab olema täpselt kolm tähte) defineerida vaid tabelis *Riik*. Tabelis *Isik* on mõistlik seda mitte teha, sest vastasel juhul on kitsendus kahes kohas ja see raskendab selle **hallatavust**.

- Veeru tüüp CHAR(3) **ei jõusta** reeglit, et veerus olev väärtus peab olema täpselt kolm tähte. Näiteks, kui lisate sellele veerule vastavasse välja ühe märgi, siis lisatakse andmebaasisüsteemi poolt selle lõppu kaks tühikut ning saadud väärtus on veeru jaoks sobiv. Sellist tüüpi veergu võib lisada nii tähti, numbreid, kirjavahemärke kui ka tühikuid. Kontrollimaks, et veerus on alati täpselt kolmest tähest koosnevad stringid e sõned, on vaja CHECK kitsendust.
- Regulaaravaldisi sisaldavad kitsendused on mõeldud vaid **tekstitüüpi** veergudele.
- Võib juhtuda, et kitsendused kattuvad selles mõttes, et A jõustamine jõustab ka B. Siis piisab vaid A jõustamisest.

Näide:
- A: {Nime esimene täht on suuräht, sellele järgneb üks või mitu väiketähte}
- B: {Nimi ei tohi olla tühi string}
- C: {Nimi ei tohi olla tühikutest koosnev string}

Jõustades A, jõustate Te ka B ja C. B ja C jaoks pole eraldi jõustamise koodi vaja.

- Koodi parema arusaadavuse ja hallatavuse huvides jõustage sarnaseid kitsendusi erinevates tabelites samal viisil.
- PostgreSQL ja Oracle lubavad viidata mittevõrduse kontrolli operaatorile kasutades nii nime != kui ka <>. Soovitan kasutada nime <>, sest sellise nimega operaator on ettenähtud SQL standardis ning standardi järgimine võib veidi parandada koodi porditavust erinevate süsteemide vahel.
  - trim(eesnimi)<>''
  - trim(eesnimi)!='' (samaväärne eelmisega, kuid != nimega operaatorit SQL standard ei kirjelda)
- Ärge jõustage üksteisega vastuolus olevaid kitsendusi. PostgreSQL ja Oracle ei oska seda tuvastada ja tulemusena ei saa lihtsalt sellesse tabelisse mingeid andmeid lisada. Näide:

```
CREATE TABLE Kontroll(kontrollitav INTEGER,
CONSTRAINT c1 CHECK (kontrollitav>0),
CONSTRAINT c2 CHECK (kontrollitav<0));
```

- Kui tahate Oracles registreerida vaid mingite sündmuste kuupäeva (kuid mitte) kellaaja, siis ärge unustage selle reegli jõustamiseks CHECK kitsendust. Oracle DATE tüüpi väärtus sisaldab ka kellaaja komponenti.

Näide:

```
CONSTRAINT chk_T990999_Koristaja_synd_kel
CHECK(trunc(synni_kp)=synni_kp)
```

- Kuidas kirjutada kitsendusse kuupäeva või ajatempli literaal? Näiteks 5. juuni 2017?
  - PostgreSQL: '2017-06-05'

Miks mitte '05.06.2017'? Kuidas PostgreSQL seda stringi tõlgendab sõltub *DateStyle* juhtparameetri väärtusest. Seda väärtust saab vaadata käsuga:

```
SHOW datestyle;
```

Tulemus *apex.ttu.ee* serveris: ISO, MDY

MDY tähendab, et Teie kuupäevastringi tõlgendus lähtub Ameerika formaadist – kõigepealt kuu identifikaator (M), siis päeva identifikaator (D) ja lõpuks aasta identifikaator (Y). Näiteks järgnevas lauses

```
CREATE TABLE Kp_test (
a DATE NOT NULL,
CONSTRAINT chk_a CHECK (a>'05.06.2017'));
```

jõustatakse kitsendus, et a väärtus peab olema suurem kui 6. mai 2017. Samas Euroopas tõlgendatakse seda stringi kui 5. juuni 2017. Segaduse vältimiseks kasutage palun ISO 8601 kuupäeva stringi formaati https://en.wikipedia.org/wiki/ISO_8601 Antud juhul oleks see '2017-06-05'. DateStyle parameetri väärtust saab ka muuta (https://stackoverflow.com/questions/13244460/how-to-change-datestyle-in-postgresql), kuid ISO formaadi kasutamine oleks ikkagi kõige selgem ja vähem segadust tekitav lahendus.

  - Oracle: to_date('2017-06-05','YYYY-MM-DD)
    - Esimene argument on kuupäeva tähistav string, teine argument on muster, mis ütleb, kuidas esimest argumenti tõlgendada.

- PostgreSQL TIMESTAMP tüüpi veerus ning Oracle TIMESTAMP tüüpi veerus saavad olla sekundi murdosad. Kui on vaja jõustada kitsendus, et suurim võimalik väärtus ei tohi olla suurem mingist kuupäevast, siis tuleb sellega arvestada. Toon näite kitsenduse loogikaavaldisest PostgreSQL põhjal. Vaja on jõustada kitsendus, et *reg_aeg* veerus olev väärtus ei tohi olla varasem kui 1. jaanuar 2010 ja hilisem kui 31. detsember 2100.
  - **Halb**: `reg_aeg BETWEEN '2010-01-01' AND '2100-12-31 23:59:59'`
    - *reg_aeg* ülapiir on kitsenduse kohaselt '2100-12-31 23:59:59**.000000**'
    - Kui *reg_aeg* on näiteks '2100-12-31 23:59:59**.20**', siis sellist registreerimise aega veergu lisada ei saa, kuigi tegemist on veel 2100. aastaga.
  - **Parem**: `reg_aeg>='2010-01-01' AND reg_aeg<'2101-01-02'`
    - Teises alamtingimuses on kirjas esimene moment peale ülapiiri möödumist (kirja pandud literaal on samaväärne kui '2101-01-02 00:00:00.000000') ning kasutusel on väiksem kui (<) operaator.
    - Kui *reg_aeg* on näiteks '2100-12-31 23:59:59**.20**', siis sellist registreerimise aega saab veergu lisada.

- Kui soovite kontrollida parooli e salasõna tugevust (nt peab olema vähemalt kuus märki, vähemalt üks suurtäht, vähemalt üks number jne), siis ei saa seda teha CHECK kitsendusega, sest parooli veergu läheb parooli ja soola põhal leitud räsiväärtus. Parooli tugevuse kontrolliks tuleb kirjutada eraldi funktsioon ja kutsuda see välja enne, kui registreeritava parooli põhjal räsiväärtust arvutama hakatakse.
- Kui veerg on tüüpi CHAR(n), siis pöörake tähelepanu järgnevale.

```
CREATE TABLE Semester (
semester_kood CHAR ( 2 ) NOT NULL,
CONSTRAINT PK_Semester PRIMARY KEY (semester_kood),
CONSTRAINT chk_Semester_kood_koosneb_kuni_kahest_tahest
CHECK (semester_kood~'^[A-Z]{1,2}$'));

INSERT INTO Semester (semester_kood) VALUES ('S');
/*ERROR: new row for relation "semester" violates check
constraint "chk_semester_kood_koosneb_kuni_kahest_tahest"
DETAIL: Failing row contains (S ).*/

DROP TABLE Semester;

CREATE TABLE Semester (
semester_kood CHAR ( 2 ) NOT NULL,
CONSTRAINT PK_Semester PRIMARY KEY (semester_kood),
CONSTRAINT chk_Semester_kood_koosneb_kuni_kahest_tahest
CHECK (semester_kood~'^[A-Z]{1}[A-Z ]{1}$'));

INSERT INTO Semester (semester_kood) VALUES ('S');
/*Lisamine õnnestus*/
```

Esimene INSERT lause ebaõnnestus, sest kuna *semester_kood* on tüüpi CHAR(2), siis lisades veergu väärtuse 'S', salvestatakse tegelikult 'S ' (CHAR tüüpi väärtuse puhul pannakse salvestatava väärtuse lõppu maksimaalse pikkuse saavutamiseks tühikud). Nimetatud string aga pole kooskõlas kitsendusega. Parandatud tabeli loomise lauses kontrollitakse, et semestri koodis oleks kõigepealt üks märk A-Z ning seejärel veel üks märk, mis on kas A-Z või tühik. Alternatiivne lahendus on deklareerida, et veerg *semestri_kood* on tüüpi VARCHAR(2). Sellisel juhul sobib esimeses CREATE TABLE lauses olev kitsendus. Näite kood on PostgreSQL jaoks, kuid samasugune probleem tekib Oracles.

- CHECK kitsenduste kirjeldamisel ärge pange kitsenduste loogikaavaldistesse (otsingutingimustesse, predikaatidesse) kokku palju lihttingimusi (atomaarseid predikaate). Loogikaavaldise näide: **`eesnimi!~'^[[:space:]]*$'`** Miks?
  - Kui avaldises on viga, siis mida pikem on avaldis, seda raskem on sealt viga leida.
  - Kui soovin ühte lihttingimust kustutada, siis ALTER TABLE ... DROP CONSTRAINT ... kustutab ka teisi, asjasse mittepuutuvaid tingimusi.
  - Kui kustutate tabelist veeru ja sellega koos kustutatakse automaatselt veerule viitavad kitsendused, siis kustutatakse ka teisi, asjasse mittepuutuvaid tingimusi.
  - Kui teete andmemuudatuse, mis läheb vastuollu kitsendusega, siis veateates näidatakse kitsenduse nime. Mida rohkem lihttingimusi on üheks kontrolliks kokku kombineeritud, seda ebaselgemaks jääb, mille vastu täpselt eksiti.

Näide:

Halvem

```
Nimi: chk_Koristaja_nimi_ei_koosne_tyhikutest
eesnimi!~'^[[:space:]]*$' AND perenimi!~'^[[:space:]]*$'
```

Parem

```
Nimi: chk_Koristaja_eesnimi_ei_koosne_tyhikutest
Avaldis: eesnimi!~'^[[:space:]]*$'

Nimi: chk_Koristaja_perenimi_ei_koosne_tyhikutest
Avaldis: perenimi!~'^[[:space:]]*$'
```

Selle põhimõtte rakendamine on huvide lahususe (*separation of concerns*) disainiprintsiibi ilming – igal kitsendusel on oma spetsiifiline ülesanne. Samal põhjusel ei looda näiteks hästi kirjutatud objektorienteeritud programmis meetodit, millel on kaks ülesannet (mille vahel valimiseks on meetodis IF/CASE laused), vaid tehakse kaks meetodit, millest kummalgi on üks ülesanne.

- Kitsenduste identifikaatorid e nimed.
  - Nimed peavad lähtuma mingist (Teie valitud) reeglistikust, mida tuleb kõigi nimede puhul **järjekindlalt** järgida. Soovitan need reeglid kuhugi kirja panna ja projekti liikmete vahel laiali jagada, et kõik neid järgiksid.
  - Eesliide või järelliide "on_korras", "on_korrektne", "on_õige" jms on sisuliselt vale, sest kitsendused ei taga andmete õigsust, vaid ainult

defineeritud reeglitele vastavuse. Kui defineeritud reeglid pole väga detailsed (nt e-meili aadress peab sisaldama @ märki), siis on palju ebakorrektseid väärtuseid, mida see kitsendus kinni ei püüa.
- Eesliide või järelliide "kontrollitud", "valideeritud" jms on (ebavajalik) müra, sest selle võiks/peaks panema siis kõikidesse nimedesse, aga samamoodi võib kõikidest nimedest ära jätta.
- Soovitan kasutada **[S/s]nake_case**.
- Tuleb jälgida, et nimed vastaksid andmebaasisüsteemi reeglitele (näiteks kõik tühikud tuleb eemaldada) ja andmebaasis kasutatavatele üldistele nimetamise reeglitele (näiteks asendada täpitähed (õÕäÄöÖüÜ), et olla kooskõlas tabelite ja veergude nimedega).

PostgreSQL: https://www.postgresql.org/docs/10/static/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS

Oracle: https://docs.oracle.com/database/121/SQLRF/sql_elements008.htm

- Tüüpviga on kasutada nimedes **tühikuid**, **@**, **-**
- Oracle puhul tuleb **kõigis** kitsenduste nimedes kasutada nimekonfliktide vältimiseks oma matrikli numbrit (nt eesliide **t990999_**). Kasutage kogu projektis läbivalt ühte matrikli numbrit.
- Kitsenduste nimed peaksid olema skeemi piires unikaalsed. Lisage kitsenduse nimesse tabeli nimi, millega see on seotud. Kui saate andmeid muutes kitsenduse vastu eksides veateate, milles sisaldub kitsenduse nimi, siis saate vea põhjuse kergemini tuvastada.
- Oracle puhul jälgige, et kitsenduste nimed poleks pikemad kui 30 baiti. Kontrollimiseks võite näiteks kasutada: https://mothereff.in/byte-counter (PostgreSQLis on *apex.ttu.ee* serveris kasutusel vaikimisi maksimaalne pikkus – **63 baiti**).

Kust leiab veel näiteid ja juhendeid regulaaravaldiste kohta?
- **Ülesandega samas kataloogis olevad failid, kus on näiteid PostgreSQL ja Oracle kohta**
- **Õppeaine näiteprojekt.**
- Regulaaravaldiste kataloog: http://www.regexlib.com/DisplayPatterns.aspx
- Jutt Oracle kohta, kuid regulaaravaldiste mustreid puudutav osa kehtib ka PostgreSQLis: http://www.dba-oracle.com/t_regular_expressions.htm
- Regulaaravaldised PostgreSQLis: https://www.postgresql.org/docs/10/static/functions-matching.html#FUNCTIONS-POSIX-REGEXP
- Regulaaravaldiste juhend: http://www.regular-expressions.info/tutorial.html

Kuidas testida regulaaravaldisi? Tuleb proovida *vähemalt* kahe tekstistringiga. Üks peab vastama regulaaravaldise mustrile ning selle kontrollimise tulemus peab olema TRUE. Teine peab olema vastuolus regulaaravaldise mustriga ning selle kontrollimise tulemus peab olema FALSE.

Võite kasutada mõnda veebipõhist vahendit: https://www.freeformatter.com/regex-tester.html

või kasutada ära andmebaasisüsteemi sisseehitatud võimet regulaaravaldisi töödelda. Veebipõhiste vahendite puhul tuleb arvestada, et erinevates keeltes/süsteemides olevad regulaaravaldiste mootorid ei pruugi olla täiesti ühilduvad, st nendes on kasutusel erinevad regulaaravaldiste dialektid (täpselt nagu SQL puhul on erinevates andmebaasisüsteemides erinevad SQL dialektid e mägimurrakud). Seega, kui kontrollite mõnda regulaaravaldist veebipõhises vahendis ja see ütleb, et avaldis ei toimi korrektselt, siis see ei tähenda tingimata, et sama tulemuse saate kui kasutate seda avaldist mõnes andmebaasisüsteemis.

Kuidas kasutada regulaaravaldiste kontrollimiseks andmebaasisüsteemi sisseehitatud funktsionaalsust?

PostgreSQL

```
SELECT '+131231'~'^([[:digit:]]|[+]|[[:space:]])+$' AS tulemus;
--Tulemus on TRUE

SELECT 'A131231'~'^([[:digit:]]|[+]|[[:space:]])+$' AS tulemus;
--Tulemus on FALSE
```

Üldkuju (PostgreSQL)

```
SELECT 'kontrollitav string'<operaator>'muster' AS tulemus;

<operaator>::= ~ või ~* või !~ või !~*
~ – vastab regulaaravaldise mustrile; tõstutundlik
~* – vastab regulaaravaldise mustrile; tõstutundetu
!~ – ei vasta regulaaravaldise mustrile; tõstutundlik
!~* – ei vasta regulaaravaldise mustrile; tõstutundetu
```

Oracle

```
DECLARE
result BOOLEAN;
BEGIN
result:=REGEXP_LIKE('+131231', '^([[:digit:]]|[+]|[[:space:]])+$');
IF (result=true) THEN
dbms_output.put_line('TRUE');
ELSE
dbms_output.put_line('FALSE');
END IF;
END;
/
--Tulemus on TRUE

DECLARE
result BOOLEAN;
BEGIN
result:=REGEXP_LIKE('A131231', '^([[:digit:]]|[+]|[[:space:]])+$');
IF (result=true) THEN
dbms_output.put_line('TRUE');
ELSE
dbms_output.put_line('FALSE');
END IF;
END;
/
```

--Tulemus on FALSE

Üldkuju (Oracle)

```
DECLARE
result BOOLEAN;
BEGIN
result:=<siia vajadusel eitus - NOT>
REGEXP_LIKE('kontrollitav string', 'regulaaravaldise
muster');
IF (result=true) THEN
dbms_output.put_line('TRUE');
ELSE
dbms_output.put_line('FALSE');
END IF;
END;
/
```

Miks ma pean Oracles kirjutama sellise *anonüümse ploki*? Sest Oracle ei võimalda väljapool PL/SQL keelt tüüpi BOOLEAN kasutada.

Selle kohta, kuidas Oracle SQL Developeris taolist anonüümset plokki käivitada ja tulemust näha, vaadake Oracle õppevideot: "Oracle SQL Developer programmis PL/SQL plokis dbms_output.put_line protseduuri abil väljastatud teate vaatamine".

Soovitan sellise ploki käivitamiseks SQLcli mitte kasutada, sest erinevalt SQL*Plusist ja SQL Developerist, annab seal sellise käivitamine veateate.

Regulaaravaldiste kirjutamisel tuleb kasutada õigeid märkide klasse, et mitte lükata tagasi andmeid, mida tegelikult ei peaks tagasi lükkama.

PostgreSQL

```
SELECT 'Õnne Pärl'~'^([a-zA-Z]|[[:space:]])+$' AS tulemus;
```

--Tulemus on FALSE

```
SELECT 'Õnne Pärl'~'^([[:alpha:]]|[[:space:]])+$' AS tulemus;
```

--Tulemus on TRUE

Oracle

```
DECLARE
result BOOLEAN;
BEGIN
result:=REGEXP_LIKE('Õnne Pärl', '^([a-zA-Z]|[[:space:]])+$');
IF (result=true) THEN
dbms_output.put_line('TRUE');
ELSE
dbms_output.put_line('FALSE');
END IF;
END;
/
```

--Tulemus on FALSE

```
DECLARE
result BOOLEAN;
BEGIN
result:=REGEXP_LIKE('Õnne Pärl', '^([[:alpha:]]|[[:space:]])+$');
IF (result=true) THEN
dbms_output.put_line('TRUE');
ELSE
dbms_output.put_line('FALSE');
END IF;
END;
/
--Tulemus on TRUE
```

Täpsemalt lugege märkide klasside kohta:
https://www.regular-expressions.info/posixbrackets.html

Antud näites kontrollitakse isiku nime. Kas oleks vaja ka enda andmebaasi selline kontroll lisada? Leian, et ei ole, sest tulenevalt kultuurilistest eripäradest võivad nimed olla väga eripalgelised:
https://shinesolutions.com/2018/01/08/falsehoods-programmers-believe-about-names-with-examples/

NB! Oracle SQL Developer (ver 18.2) kasutamisel ilmnes järgnev probleem. Meil on tegemist regulaaravaldisega riikide ISO klassifikaatori koodide kontrollimiseks, mille kohaselt peab kood koosnema täpselt kolmest suurtähest.

```
DECLARE
result BOOLEAN;
BEGIN
result:=REGEXP_LIKE('EST', '^[A-Z]{3}$');
IF (result=true) THEN
dbms_output.put_line('TRUE');
ELSE
dbms_output.put_line('FALSE');
END IF;
END;
/
```

Oracle APEX, SQL*Plusis ja SQLclis annab see test tulemuseks **TRUE**, Oracle SQL Developeris (ver 18.2) annab see tulemuseks **FALSE.**

Kui tabelis on selline CHECK kitsendus (nt veerul riik_kood), siis Oracle APEXis ja SQL Plusis õnnestub sellise CHECK kitsendusega tabelisse kenasti andmeid lisada. Samas SQL Developeris öeldakse andmete lisamisel (nt riik koodiga EST), et CHECK kitsenduse viga – seda nii INSERT lauseid käivitades, kui otse tabelisse andmeid lisades.

Soovi korral võite kasutada regulaaravaldise mustrit **'^[[:upper:]]{3}$'**, mis toimib tõrgeteta kõigis minu kasutatud Oracle andmebaasisüsteemiga suhtlemiseks mõeldud keskkondades.

Pange tähele, et mustrid **'^[[:upper:]]{3}$'** ja **'^[A-Z]{3}$'** pole samaväärsed, sest esimene lubab ka täpitähti (ÕÄÖÜ), kuid teine mitte. Kuna riikide koodides pole selliseid täpitähti, siis oleks mustrina eelistatud '^[A-Z]{3}$'.

PostgreSQLis toimib regulaaravaldis '^[A-Z]{3}$' nii nagu peab.

Indeksid

Välisvõtmetele loodavate indeksite kirjeldamise alustamiseks vaadake eeldusaine kodulehelt videot "Andmebaasi diagrammis indeksite kirjeldamine".

Peate lisama indeksid välisvõtme veergudele (ka hindamismudel nõuab seda). Muude veergude indekseerimine on vabatahtlik. Kuna indeksid aeglustavad andmete muutmist, siis operatiivandmete andmebaasis on indekseid pigem vähem kui rohkem.

Millele välisvõtmetele loodavate indeksite kirjeldamisel tähelepanu pöörata?

- PostgreSQL ja Oracle loovad automaatselt indeksi PRIMARY KEY ja UNIQUE kitsenduse alusel. Välisvõtmeid need süsteemid automaatselt ei indekseeri.
- Välisvõtmetele indeksi loomisel tuleb jälgida, et loodav indeks ei dubleeriks olemasolevaid (automaatselt loodud) indekseid.
- Kui tegite generaatoris õigeid valikuid, siis võivad loodud andmebaasi disaini mudelis kõik või osa välisvõtmetele mõeldud indekseid olla juba kirjeldatud. Teie ülesandeks on järgnevat arvesse võttes need indeksid üle vaadata ning vajadusel indeksite kirjeldusi juurde lisada, muuta või kustutada.

Millal välisvõtit mitte indekseerida?

- Tellimuse_rida(tellimus_id, kaup_id, kogus)
  Primaarvõti (tellimus_id, kaup_id)
  Välisvõti (**tellimus_id**)
  Viitab Tellimus(tellimus_id)
  Välisvõti (**kaup_id**) Viitab Kaup(kaup_id)

- Enamik andmebaasisüsteeme loob automaatselt **liitindeksi** (tellimus_id, kaup_id):
  - tellimus_id – esimene veerg liitindeksis,
  - saab alati kasutada, kui on vaja leida konkreetse tellimuse read.
- Järelikult mitte luua täiendavat indeksit (tellimus_id)!!
- Tuleb luua indeks (kaup_id).

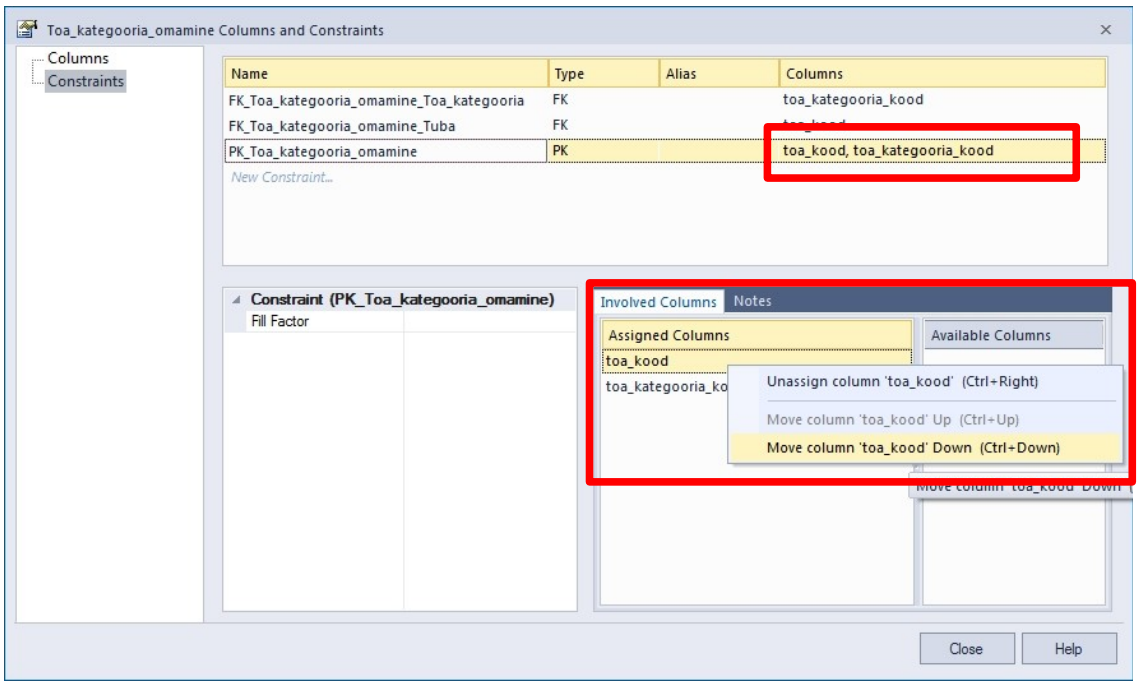
Miks on vaja *kaup_id* täiendavalt indekseerida?

Ülo Mets Valga	1184
Hannes Mets Inju, Vinni Vald, Lääne-Viru Maakond	1184
Ergo Mets Sõstra, Rakvere	1184
Martin Mets Kalda, Kadrina, Kadrina Vald, Lääne-Viru Maakond	1184
Natalje Mets Ravi 4, Väike-Maarja, Väike-Maarja Vald, Lääne-Viru Maakond	1184
Henno Mets Roosi, Roela, Vinni Vald, Lääne-Viru Maakond	1184
Virve Mets Puru Tee 24, Jõhvi Vald, Jõhvi	1184
Vilma Mets Liiva, Toila, Toila Vald, Ida-Viru Maakond	1184
Üllar Mets Pühaoru 9, Toila, Toila Vald, Ida-Viru Maakond	1184
Linda Mets Pikk, Toila, Toila Vald, Ida-Viru Maakond	1184

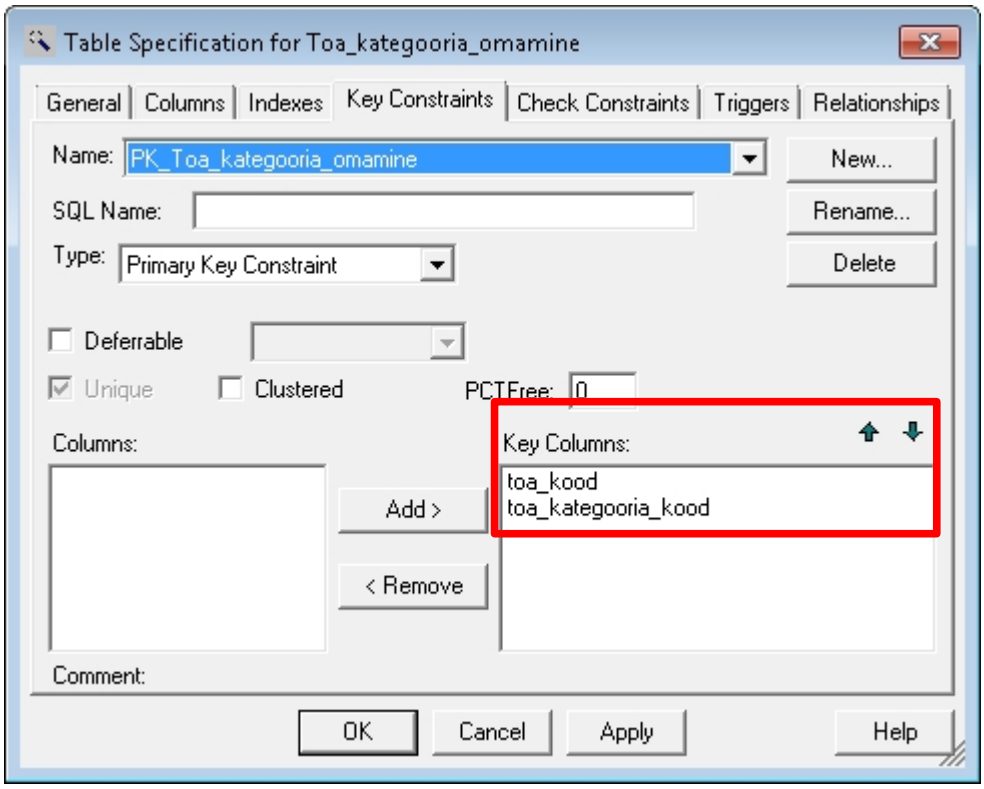
- Analoogiaks telefoniraamat
  - Liitindeks (perenimi, eesnimi).
  - Kui otsin isikut perenime või perenime + eesnime järgi, siis on otsimine kiire, sest isikute andmed on perenime ja selle sees eesnime järgi sorteeritud.
  - Kui otsin isikuid **ainult** eesnime järgi, siis pean vaatama läbi kogu raamatu.
    - antud näites eesnimi = kaup_id

Kuidas teha CASE vahendis kindlaks, milline veerg on liitvõtmes esimene veerg ning kuidas seda järjekorda muuta? Järgnevas näites on *toa_kood* liitvõtme esimene veerg.

EA12 – veergude järjekorra muutmiseks tuleb muuta *Assigned Columns* järjekorda (kontekstitundlikust menüüst, mis avaneb parema hiire klahvi alt).



RR – veergude järjekorra muutmiseks tuleb muuta *Key Columns* järjekorda (noole pildiga nupud).



Välisvõtmete indekseerimise algoritmi pseudokood:

```
foreach välisvõtme veerg v {
              if v ei ole esimene veerg üheski PRIMARY KEY
              või UNIQUE kitsenduses {
                    Loo indeks veerule v
              }
}
```

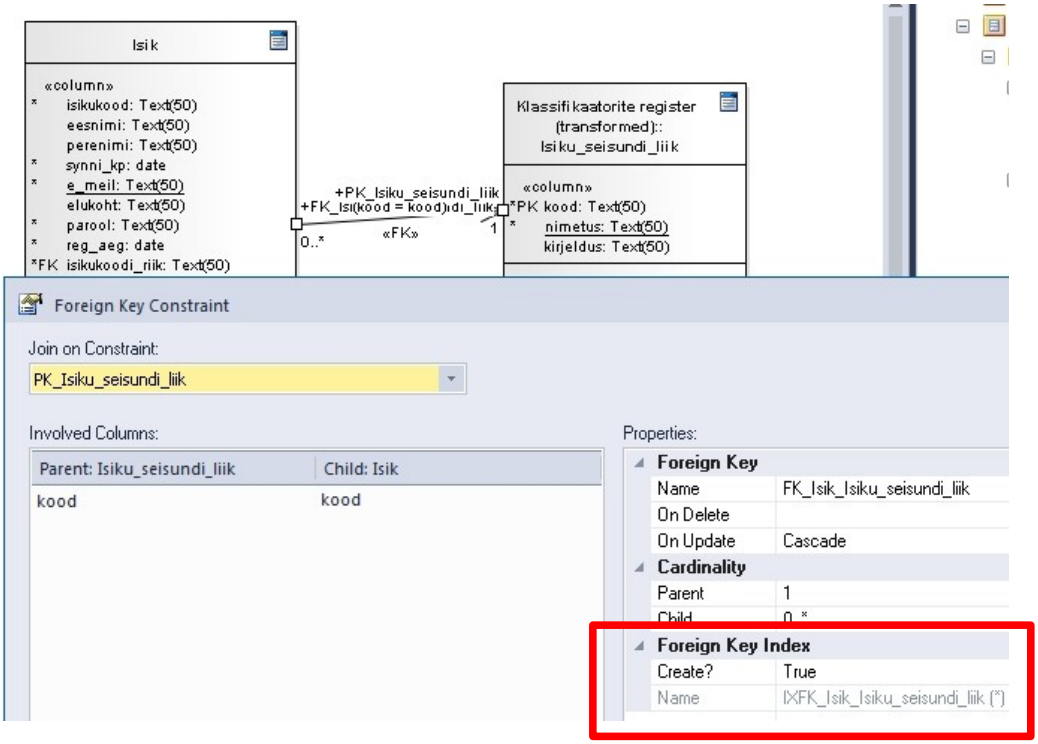
- Mõelge läbi, milline on mitut veergu hõlmavate PRIMARY KEY ja UNIQUE kitsenduste korral veergude järjekord (hõlmaku need kitsendused välisvõtme veerge või mitte). Näiteks, kui defineerida tabelis *Autor* unikaalsuse kitsendus UNIQUE (perenimi, eesnimi, sünniaasta, kommentaar), siis loob andmebaasisüsteem selle alusel liitindeksi, mis hõlmab samu veerge ja kus veerud on samas järjekorras. Andmebaasisüsteem võib kaaluda sellise indeksi kasutamist päringu kiiremaks täitmiseks, kui üks päringu WHERE klauslis olev tingimus puudutab perenime (liitindeksi esimene veerg) – näiteks otsing perenime, perenime + eesnime või perenime + eesnime + sünniaasta järgi. Samas ei saa andmebaasisüsteem seda indeksit kasutada, kui otsingu tingimuses perenimele ei viidata – nt otsing eesnime või eesnime + sünniaasta järgi. Mõelge jälle telefoniraamatule, kus isikud on sorteeritud perenime järgi. Kuni isikut otsitakse perenime või perenime + eesnime järgi on sellest järjekorrast palju abi. Kui otsing käib ainult eesnime järgi, siis tuleb kogu telefoniraamat läbi vaadata. Valige

liitvõtmetes selline veergude järjekord, et esimest veergu kasutatakse nendest kõige sagedamini päringu tingimustes.
- Indeksite nimetamisel jälgige samu asju nagu CHECK kitsenduste nimetamisel.

Mõelge ka sellele, kas Teie tabelites on veel mõni indekseerimata veerg, mida kasutatakse sageli käivitatavate päringute tingimustes. Indeksid muudavad aeglasemaks tabelisse ridade lisamise ning indekseeritud veergudes andmete muutmise, kuid kiirendavad osasid päringuid, mille tingimuses viidatakse indekseeritud veerule. Kokkuvõttes tuleb läbi mõelda, kas oleks kasulik mõni selline veerg ikkagi indekseerida. Võiksite seda teha juba disaini mudelis.

EA kasutamise korral pöörake tähelepanu, et kui vajutate välisvõtme kirjeldamise aknas OK, siis võib see tingida välisvõtme veerule mõeldud indeksi automaatse kirjeldamise. Indeksi nimi tuletatakse välisvõtme kitsenduse nimest. Ühelt poolt on see hea võimalus puuduvate indeksite kiireks kirjeldamiseks.

Teisalt olge palun tähelepanelik, sest kui olete mõne indeksi kirjelduse teadlikult eemaldanud ning vajutate vastava välisvõtme kirjelduse aknas OK (kusjuures Foreign Key Index Create? = True), siis tekib selle indeksi kirjeldus tagasi.



Milliste veergude indekseerimist võiks veel kaaluda?
- Liitindeks (perenimi, eesnimi) tabelis *Isik*, et kiirendada isikute otsimist nime järgi. Kui *perenimi* on selle indeksi esimene veerg, siis saab andmebaasisüsteem kaaluda indeksi kasutamist, kui isiku otsing tehakse perenime või perenime + eesnime järgi.
- Osaline indeks tabeli *Klient* veerule *on_nous_tylitamisega*. Selle indeksi võiks luua ridadele, kus *on_nous_tylitamisega=TRUE*, sest just selliseid ridu otsivad ilmselt otseturunduse tegijad. Sellist indeksit ei saa EA ning

RR vahendis kirjeldada. Seega tuleb sellise indeksi loomise kood ise kirjutada.
  - PostgreSQL: https://www.postgresql.org/docs/10/static/indexes-partial.html
  - Oracle: http://dba-presents.com/index.php/databases/oracle/41-filtered-index-equivalent-in-oracle
- PostgreSQL korral BRIN (plokkide vahemiku) indeksid veergudel *reg_aeg*. Read on registreerimisaja järgi kettal loomulikul viisil enam-vähem järjestatud ja loodud indeksist oleks abi, kui ridu otsitakse registreerimisaja vahemike alusel. Sellist indeksi tüüpi ei saa EA ning RR vahendis kirjeldada. Seega tuleb sellise indeksi loomise kood ise kirjutada.
  - https://sweetness.hmmz.org/2015-05-22-block-range-brin-indexes-in-postgresql-95.html
  - https://www.postgresql.org/docs/10/static/brin-intro.html

Selle ülesande lahendamiseks saab lisainfot slaidikomplektist:
- **Baastabelid_systeemikataloog_IDU0230_2018.ppt**

, mille leiab http://193.40.244.90/346