

## Ülesanne 4

Kui mõtlete, et ülesannetes 2 ja 3 oli liiga palju detaile ja pisiasju, millele tähelepanu pöörata, siis lisaks *tehnilisest võlast* lahtisaamisele on nende kõikide ülesannete juures oluline ka "**katkiste akende teooria**" (<https://blog.codinghorror.com/the-broken-window-theory/>).

See kriminoloogia teooria väidab, et kui taluda mingis piirkonnas kergemaid väärtegusid (nt mõne akna katki viskamine või grafiti), siis julgustab see tõsisematele õigusrikkumistele ja piirkond käib kiiresti alla. Isegi tavaolukorras seadusekuulekad kodanikud hakkavad sellises ümbruses halvasti käituma. Kindlasti olete isegi näinud – keegi viskab veidi prahti maha ja peagi laiutab seal kohal pisikene prügimägi. Ilmselt kehtib see teooria ka ümberpööratult – kui mingi piirkond on korras ja hoolitsetud, siis ei taha keegi olla esimene kes selle ära reostab.

Täpselt sama probleem esineb tarkvaraarenduses. Ka pisikesi probleeme tarkvaras ei tohiks taluda ja need tuleks võimalikult kiiresti kõrvaldada. Vastasel juhul käib tarkvara kiiresti alla. Selle kohta öeldakse, et tarkvara roiskub ([https://en.wikipedia.org/wiki/Software\\_rot](https://en.wikipedia.org/wiki/Software_rot)). Lohakus tõmbab ligi lohakust, vead uusi vigu. Tarkvaraarenduses osalejad vaatavad, et tarkvaras on probleeme, võtavad sellest eeskuju ja sellega tuleb neid probleeme aina juurde.

**Mis peab olema selle ülesande läbimise tulemusena tehtud?**

Parent: Koristaja	Child: Koristamine
koristaja_ID	koristaja_ID

Properties:	
<b>Foreign Key</b>	
Name	FK_Koristamine_Koristaja
On Delete	
On Update	
<b>Cardinality</b>	
Parent	1
Child	0..*
<b>Foreign Key Index</b>	
Create?	True
Name	IXFK_Koristamine_Koristaja (*)

☒ Automatically show this screen when tables are joined

Buttons: Delete, OK, Cancel, Help

- ▲ Kui kasutate **Enterprise Architecti (EA)**, siis olete üle vaadanud kõigi välisvõtme kitsenduste kirjeldused, kas need ikka on õiged – kas on õigesti määratud nii primaartabeli kui sõltuva tabeli poolt osalevad veerud. Elu näitab, et EAs võivad selles osas kergesti tekkida vead ja isegi niimoodi, et

diagrammil paistab justkui kõik õige, aga mudelis on kirjeldus ebaõige. Kui siin on vigu, siis kas välisvõtme kitsendust ei genereerita või genereeritakse see valesti.

Järgnevalt on näidatud, kuidas näeb välja "katkine" välisvõtme kirjeldus EA11 ja EA12.

Foreign Key Constraint

Name: FK\_Liige\_Ylioplane ☒ Override Template

Source: Liige Target: Ylioplane

Key	Column	Type
PK	matrikkel	char
	liikme_seisu...	smallint
	liikme_staa...	smallint
	parool	varchar

Key	Column	Type
PK	matrikkel	char
	eriala_liik_kood	char
	omandatav_kra...	smallint
	eesnimi	varchar
	perenimi	varchar

Source Cardinality: 0..1 Target Cardinality: 1

Referential Integrity

On Delete: Cascade On Update: Cascade

☐ Create Index on Foreign Key

Apply Delete

Column Type

matrikkel varchar

OK Cancel Help

Foreign Key Constraint

Join on Constraint: PK\_Koristaja

Involved Columns:

Parent	Child
Koristaja	Koristamine

koristaja\_id

Properties:

**Foreign Key**

Name: FK\_Koristamine\_Koristaja

On Delete: On Update:

**Cardinality**

Parent: 1 Child: 0..\*

**Foreign Key Index**

Create?: True Name: IXFK\_Koristamine\_Koristaja (\*)

☒ Automatically show this screen when tables are joined

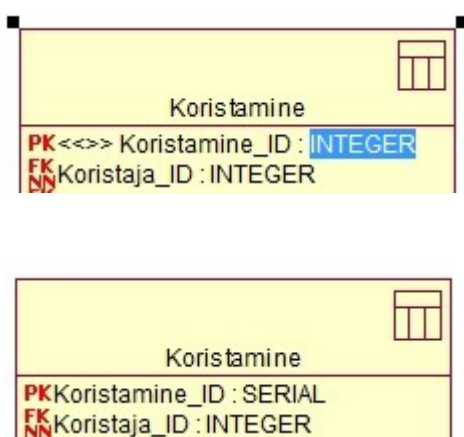
Delete OK Cancel Help

Kuidas seda parandada kirjeldatakse siin: [http://193.40.244.90/helpdesk.php?kysimus=459&kysimuse\\_teema=98&aine=369](http://193.40.244.90/helpdesk.php?kysimus=459&kysimuse_teema=98&aine=369)

Kui parandamine ei õnnestu, siis jätke seos nii nagu see on (see on CASE vahendi kala ning selle tõttu liigne aja kulutamine on raiskamine) ja hoolitsege, et genereeritud koodis ja selle alusel loodud andmebaasis oleks seos õigel viisil ja õigete omadustega deklareeritud (muutke genereeritud koodi).

- ⤴ Olete teinud mudelis kõik võimaliku, et andmebaasikeele lausetes oleks realiseeritud (loomulikult ainult seal kus vaja) surrogaatvõtmete genereerimine. Kõike pole võimalik mudelis teha ja seega tuleb olla valmis koodi käsitsi muutmiseks.

Kui teete projekti **PostgreSQLis** ja kasutate **Rational Rose'i (RR)**, siis saab diagrammile kirjutada tüübi nime (nt SERIAL, BOOLEAN, TEXT), mida veeru kirjeldamisel liitboksist valida ei saa. See nimi ilmub ka genereeritud koodi.



Kui teete projekti **PostgreSQLis** ja kasutate **EAd**, siis saab SMALLSERIAL/SERIAL/BIGSERIAL notatsiooni valida veeru kirjelduse juures liitboksist. Veenduge, et kui primaarvõtme veerg on SMALLSERIAL/ SERIAL/ BIGSERIAL, siis sellele viitav välisvõtme veerg on SMALLINT/INTEGER/ BIGINT. EA tekitab siinkohal vigu – näiteks määrade, et primaarvõti on SERIAL ja seotud välisvõti määratakse ka automaatselt SERIAL'iks. See on vale, sest välisvõtmesse ei genereeri väärtuseid süsteem vaid sinna tuleb valida väärtus seotud võtme väärtuste hulgast.

Kui teete projekti **Oracles**, siis tuleb surrogaatvõtmete väärtuse genereerimise funktsionaalsus lisada genereeritud koodi.

- ⤴ Olete veelkord mõelnud andmetüüpide valikule. Nagu eelnevast näitest näha, siis RRis saab veeru kirjeldamisel valikust puuduva tüübi nime kirjutada diagrammile. EAs saab tüübi veeru kirjeldamisel liitboksist valida, kuna EA pakub nii PostgreSQL kui Oracle tüüpe.
  - PostgreSQLis on võimalik kasutada veeru andmetüübina tüüpi BOOLEAN. Oracles pole võimalik kasutada BOOLEAN tüüpi veeru tüübina ja seal peaks kasutama selleks puhuks tüüpi NUMBER(1) koos kitsendusega, et vastavas veerus on lubatud väärtused 0 ja 1.

- Soovitan määrata BOOLEAN tüüpi veerule vaikimisi väärtuse TRUE või FALSE ja teha seda juba disaini mudelis. Veeru *on\_nous\_tylitamise*ga korral oleks korrektne vaikimisi väärtus FALSE, et isik peaks ise nõustuma.
- Kui soovite maksimaalset võimalikku tekstivälja suurust, siis PostgreSQLis on tüüp TEXT ning Oracles CLOB.
- PostgreSQLis tuleks tüüpi MONEY asemel kasutada tüüpi DECIMAL (sünonüüm NUMERIC) koos sobiva täpsusega (maksimaalne numbrikohtade arv) ja skaalaga (kohtade arv peale koma).
- Kui soovite hoida PostgreSQL andmebaasis faile (mitte viiteid asukohtadele väljapool andmebaasi), siis lugege: <https://www.postgresql.org/docs/11/static/lo-funcs.html>
- Kui soovite hoida PostgreSQL andmebaasis geograafilisi koordinaate, siis kaaluge tüüpi *Point* kasutamist:
  - <https://www.postgresql.org/docs/11/static/datatype-geometric.html>
  - <https://stackoverflow.com/questions/1023229/spatial-data-in-postgresql>
  - <https://stackoverflow.com/questions/18491802/postgresql-select-query-to-extract-latitude-and-longitude-from-a-point>
- PostgreSQL ja Oracle võimaldavad kasutada massiivitüüpe: PostgreSQL: <https://www.postgresql.org/docs/11/static/arrays.html> Oracle: [http://www.dba-oracle.com/tips\\_oracle\\_varray.htm](http://www.dba-oracle.com/tips_oracle_varray.htm) ning PostgreSQLis saab luua ja kasutada loendustüüpe: <https://www.postgresql.org/docs/11/static/datatype-enum.html> Soovitan loendustüübi kasutamist kaaluda *vaid siis*, kui olete täiesti veendunud, et sellesse tüüpi kuuluvate väärtuste hulk ajas ei muutu (reaalsuses harv juhused).
- ⤴ Olete genereerinud CASE vahendis loodud disaini mudelist andmekirjelduskeele laused tabelite, deklaratiivsete kitsenduste ning indeksite loomiseks (CREATE, ALTER) ja kustutamiseks (DROP). Seda, kuidas genereerimist läbi viia, vaadake palun videotest.

Selleks, et EA12s olevast mudelist genereeritud koodis ei oleks piiritletud identifikaatoreid (jutmärkides nimesid), on võimalik teha muudatus koodi genereerimise aluseks olevas mallis:

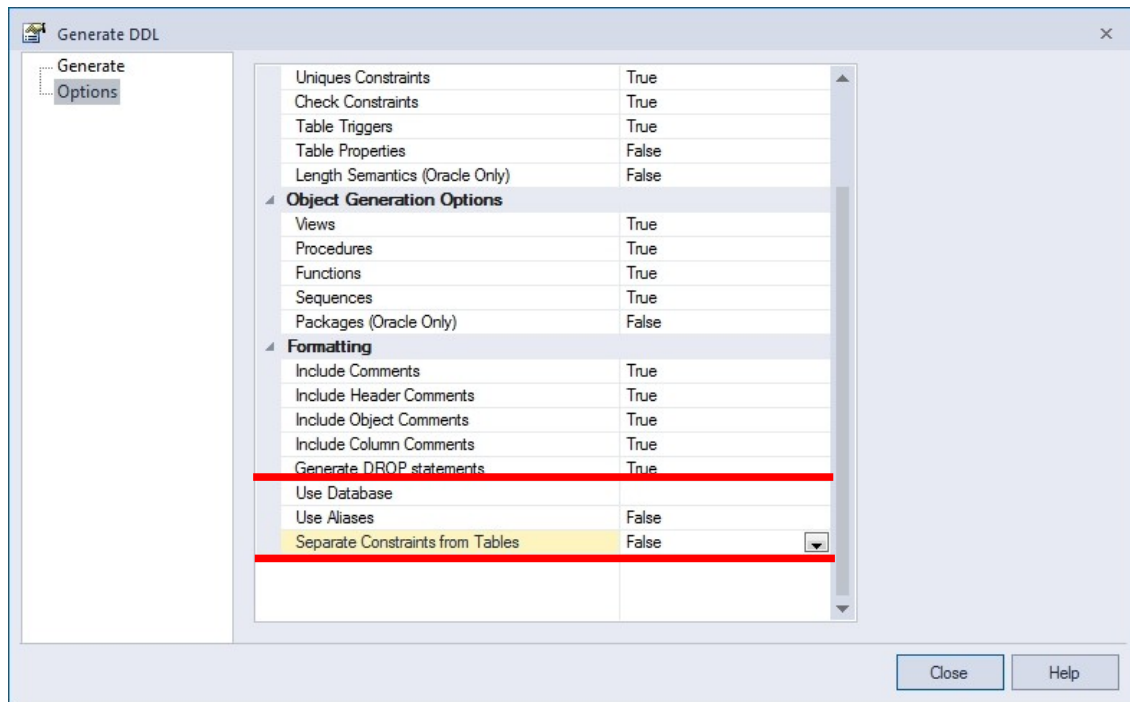
<https://stackoverflow.com/questions/37111074/enterprise-architect-generate-sql-without-quotes-in-table-name/46910049#46910049>

Oma arvutis on Teil kindlasti malli muutmise õigus. EA13 puhul genereeritakse nimed vaikimisi jutumärkideta.

**NB!** Kui kasutate EA 12, siis aktiveerige tabelite kirjeldust sisaldav pakett ning valige:

Package => Database Engineering => Generate Package DDL ...

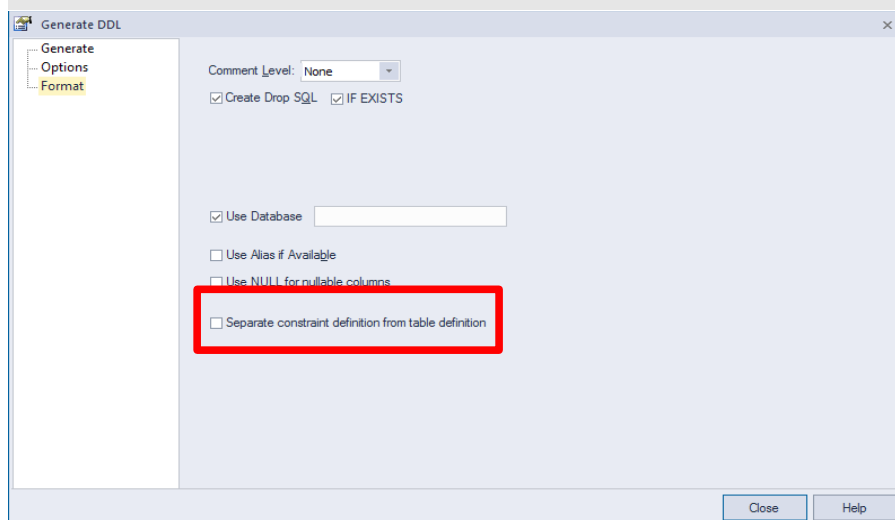
Vajadusel valige *Include all Child Packages*. **Options** alt määrake kindlasti järgnev.



***Separate Constraints from Tables = True*** annab tulemuseks koodi, kus eraldi on tabelite loomise laused (CREATE TABLE) ja kitsenduste lisamise laused (ALTER TABLE). Selline kood on õppejõu hinnangul väga halvasti loetav.

EA11 sellist valikut teha ei saa. Kui kasutate oma arvutis EA11, siis kui vähegi võimalik, genereerige kood arvutiklassi arvutis kasutades EA12, et saaksite sellise valiku teha. Seal on EA12 uuem *build* (järk), mis seda teha võimaldab.

Kui kasutate EA12 vanemat *buildi* (järku), siis näete järgnevat valikut. Sellisel juhul genereerige kood samuti arvutiklassi arvutis.



Põhjus on selles, et see EA12 genereerib Teile *ebakorrektsed* tabelite ja indeksite loomise laused.

```
CREATE TABLE Treening
(
    treeningu_kood char(3) NOT NULL,
    nimetus varchar(50) NOT NULL,
    isik_id integer NOT NULL,
    treeningu_seisundi_liik_kood smallint NOT NULL DEFAULT 1,
    spordiala_kood char(6) NOT NULL,
    kestvus smallint NOT NULL,
    kirjeldus varchar(255) NOT NULL,
    kohad smallint NOT NULL,
    nadalatunnid smallint NOT NULL,
    reg_aeg timestamp with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP(0),
    CONSTRAINT PK_Treening PRIMARY KEY (treeningu_kood),
    CONSTRAINT FK_Treening_seisundi_liik FOREIGN KEY (treeningu_seisundi_liik_kood) REFERENCES Treeningu_seisundi_liik (Treeningu_seisundi_liik_kood),
    CREATE INDEX IXFK_Treening_treeningu_seisundi_liik (treeningu_seisundi_liik_kood ASC),
    CREATE INDEX IXFK_Treening_tootaja (isik_id ASC),
    CREATE INDEX IXFK_Treening_spordiala (Spordiala_kood ASC),
    CONSTRAINT chk_Treening_nimetus CHECK (nimetus <> '' AND NOT (nimetus LIKE '% ')),
    CONSTRAINT chk_Treening_reg_aeg CHECK (reg_aeg BETWEEN '2010-01-01 00:00' AND '2100-12-31 23:59'),
    CONSTRAINT chk_Treening_kestvus CHECK (kestvus BETWEEN 30 AND 90),
    CONSTRAINT chk_Treening_kohad CHECK (kohad BETWEEN 1 AND 25),
    CONSTRAINT chk_Treening_nadalatunnid CHECK (nadalatunnid BETWEEN 1 AND 15),
    CONSTRAINT FK_Treening_spordiala FOREIGN KEY (Spordiala_kood) REFERENCES Spordiala (Spordiala_kood) ON DELETE No Action ON UPDATE Cascade,
    CONSTRAINT FK_Treening_treeningu_seisundi_liik FOREIGN KEY (treeningu_seisundi_liik_kood) REFERENCES Treeningu_seisundi_liik (Treeningu_seisundi_liik_kood) ON DELETE No Action ON UPDATE No Action,
    CONSTRAINT FK_Treening_tootaja FOREIGN KEY (isik_id) REFERENCES Tootaja (isik_id) ON DELETE No Action ON UPDATE No Action
)
```

CREATE INDEX laused on eraldi laused, mitte üks osa CREATE TABLE lausetest.

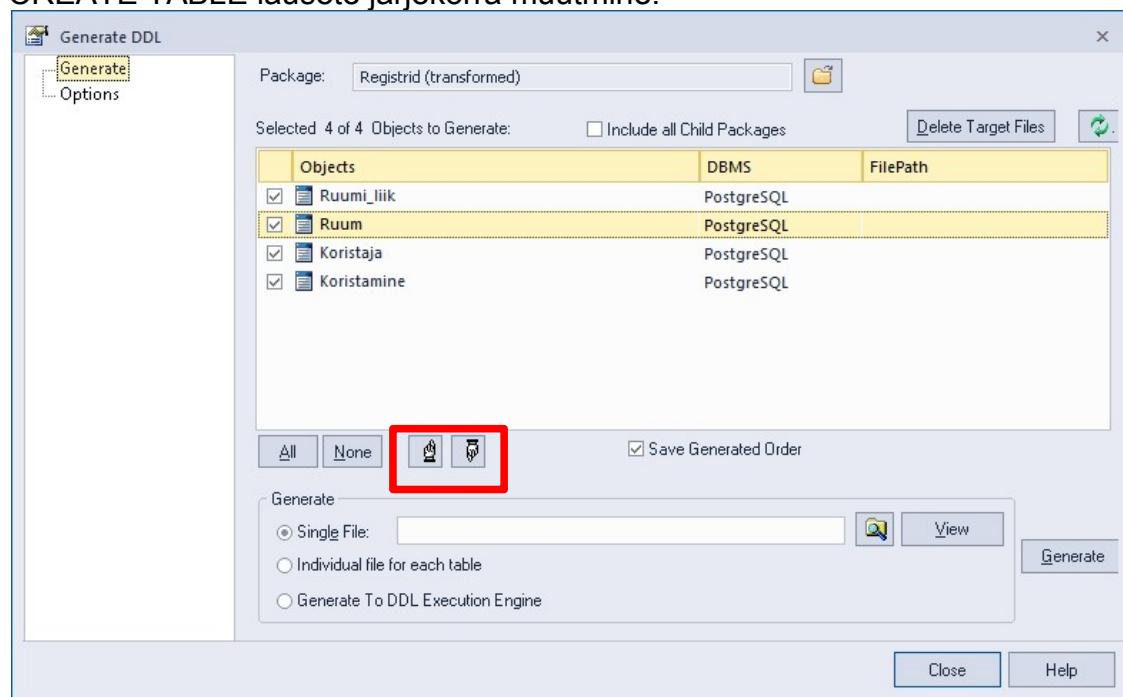
PostgreSQL – CREATE INDEX

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

Oracle – CREATE INDEX

[https://docs.oracle.com/database/121/SQLRF/statements\\_5013.htm](https://docs.oracle.com/database/121/SQLRF/statements_5013.htm)

CREATE TABLE lausete järjekorra muutmine.



Kuna EAs sellisel viisil genereeritud tabeli loomise lausetes sisalduvad välisvõtmete deklaratsioonid, mis võivad viidata teistele tabelitele (st CREATE TABLE laused hakkavad üksteisest sõltuma), siis on oluline lausete käivitamise

järjekord. Kui lauses viidatakse teistele tabelitele, siis lause täitmise õnnestumiseks peavad viidatavad tabelid andmebaasis olemas olema. Saate tabelite loomise lausete järjekorda muuta genereeritud koodis (halvem, sest koodi uuesti genereerimisel tuleb järjekorda uuesti parandada) või mudelis (parem, sest järjekorra saab salvestada)(vt joonis).

Õige tabelite loomise lausete järjekord on selline, et kõigepealt peaksid tulema ilma välisvõtmeteta tabelid ja seejärel riburada välisvõtmetega tabelid.

Töövihiku projekti korral oleks järjekord selline:

- ▲ Klassifikaatorite tabelid (X\_kategooria\_type enne X\_kategooria)
- ▲ Isik
- ▲ Klient, Tootaja
- ▲ X
- ▲ X\_kategooria\_omamine, muud X-st sõltuvad tabelid

Kui olete koodi valmis genereerinud, siis avage see tekstiredaktoris ning vaadake, kui suure osa sellest moodustavad sellest erinevad identifikaatorid (tabelite, veergude, kitsenduste, indeksite nimed). Vaadake seda ja laske endas settida arusaamal, et nende nimede selgus, järjekindlus, täpsus aitab väga palju kaasa sellele, et seda koodi hästi mõistetakse. Kui nimed pole selged, järjekindlad, täpselt sisu kirjeldavad, siis on ka koodi raske mõista ja seda hiljem edasi arendada.

- ▲ Olete laused ülevaadanud, parandanud ja täiendanud.
  - Genereerige CASE vahendi abil fail, mille nimi lõpeb **sql** laiendiga. Peale lausete genereerimist avage neid sisaldav fail *SciTe* või *Notepad+* programmi abil. Need programmid oskavad koodi värvida (näiteks võtmesõnu ja literaale esile tõsta), näitavad ridade numbreid ning selle abil saab koodist hea visuaalse ülevaate. **Mida täpselt on vaja üle vaadata, parandada ja täiendada, seda lugege palun alates lk 12.**
- ▲ Valikuline! Olete *soovi korral* täiendanud genereeritud lauseid nii, et neid saaks käivitada ühe loogilise tervikuna, st täidetakse kas kõik laused või kasvõi ühe vea korral jäetakse kõik täitmata. **Kui ebaõnnestub ühe tabeli loomine, siis hakkavad kaskaadselt ebaõnnestuma ka järgmised laused, mis viitavad loomata jäänud andmebaasiobjektidele.** Lausete loogilise tervikuna käivitamisel väldite hüpoteetilist olukorda, et tõmbate laused korraga käima ja näiteks kaheksa lauset täidetakse ning üheksa jääb täitmata. Pärast kulub hulk aega ja energiat, et nende kaheksa lause tulemused andmebaasist eemaldada, et puhtalt lehelt alustada. Alternatiiv on panna lauseid tööle ükshaaval (vt üllesande järgmine punkt).
  - PostgreSQLis tuleb selleks koondada laused ühte transaktsiooni e tehingusse. See on võimalik, sest erinevalt Oraclest lubab PostgreSQL andmekirjelduskeele lauseid transaktsiooni koondada ja rullib vähemalt ühe lause ebaõnnestumisel kogu transaktsiooni tagasi. Täiendage skriptifaili nii, et kõige esimene lause on START TRANSACTION; ja kõige viimane lause on COMMIT; Kui vähemalt ühe andmekirjelduskeele lause täitmine ebaõnnestub, siis vaatamata COMMIT lausele terve transaktsioon ebaõnnestub. Skripti käivitamise tulemus sõltub vahendist, mille kaudu seda teete.



- psqlis jooksutatakse kogu skript läbi. Peale ühe lause ebaõnnestumist ignoreeritakse kuni COMMIT lauseni kõiki järgnevaid lauseid. Skripti lõpetava COMMIT lause tulemusel rullitakse transaktsioon ikkagi tagasi.

psqli nõrkus on, et kui käivitatavas skriptis ei ole sulud või apostroofid tasakaalus (mõni on puudu/üleliia), siis see muudab psqli sessiooni teovõimetuks. Selle sümptomiks on psqli prompti e viiba muutus.

#### Näide 1 (apostroof on puudu)

```
erki=# START TRANSACTION;
START TRANSACTION
erki=#
erki=# CREATE TABLE U(u_id INTEGER PRIMARY KEY, tekst TEXT
erki(# CONSTRAINT chk_u_tekst CHECK(tekst~^[[:space:]]*$',));
erki'#
erki'# CREATE TABLE V(v_id INTEGER PRIMARY KEY);
erki'#
erki'# COMMIT;
erki'#
```

#### Näide 2 (sulg on puudu)

```
START TRANSACTION
erki=#
erki=# CREATE TABLE U(u_id INTEGER PRIMARY KEY;
erki(#
erki(# CREATE TABLE V(v_id INTEGER PRIMARY KEY);
erki(#
erki(# COMMIT;
erki(#
```

Esimene lahendus on sisestada psqli käsitsi märke, kuni viga saab ravitud ning parandustega koodi saab taaskäivitada.

#### "Ravi" esimese näite korral

```
erki'# '
erki(# )
erki(# )
erki-# ;
ERROR: syntax error at or near "["
LINE 2: CONSTRAINT chk_u_tekst CHECK(tekst~^[[:space:]]*$',));
                                         ^

erki=# ROLLBACK;
ROLLBACK
erki=#
```

#### "Ravi" teise näite korral

```
erki(# )
erki-# ;
ERROR: syntax error at or near ";"
LINE 1: CREATE TABLE U(u_id INTEGER PRIMARY KEY;
                                         ^

erki=# ROLLBACK;
ROLLBACK
erki=#
```

Teine lahendus on sessioon jõuga lõpetada (antud juhul PuTTY aken sulgeda) ning uuesti alustada.



- pgAdminis jääb skripti täitmine esimese vea tekitanud lause juures pooleli. Ka transaktsioon jääb pooleli. Jätkamiseks tuleb transaktsioon kõigepealt ROLLBACK lausega tagasi rullida ja siis uuesti alustada. Selleks tuleb aknast kõik käsud kustutada, käivitada ROLLBACK lause ning seejärel parandustega skript tagasi kleepida.

```

START TRANSACTION
erki=#
erki=# CREATE TABLE U(u_id INTEGER PRIMARY KEY);
CREATE TABLE
erki=#
erki=# CREATE TABLE V(v_id INTEGR PRIMARY KEY);
ERROR:  type "integr" does not exist
LINE 1: CREATE TABLE V(v_id INTEGR PRIMARY KEY);
                                ^

erki=#
erki=# CREATE TABLE X(x_id INTEGER PRIMARY KEY);
ERROR:  current transaction is aborted, commands ignored until end
of transaction block
erki=#
erki=# COMMIT;
ROLLBACK
erki=#

```

```

1  START TRANSACTION;
2
3  CREATE TABLE U(u_id INTEGER PRIMARY KEY);
4
5  CREATE TABLE V(v_id INTEGR PRIMARY KEY);
6
7  CREATE TABLE X(x_id INTEGER PRIMARY KEY);
8
9  COMMIT;

```

Data Output Explain Messages Query History

```

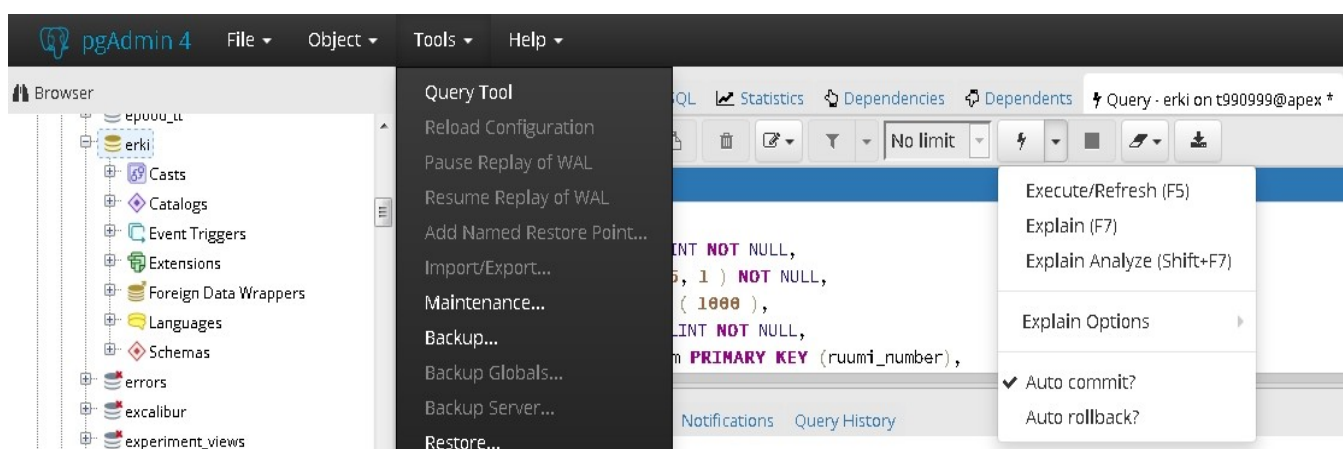
ERROR:  type "integr" does not exist
LINE 5: CREATE TABLE V(v_id INTEGR PRIMARY KEY);
                                ^

SQL state: 42704
Character: 84

```

- Oracle korral tuleb mitme CREATE TABLE lause loogilise tervikuna täitmise saavutamiseks need laused koondada CREATE SCHEMA lausesse.
  - [https://docs.oracle.com/database/121/SQLRF/statements\\_6016.htm](https://docs.oracle.com/database/121/SQLRF/statements_6016.htm)
- ^ Olete tabelite, kitsenduste ning indeksite loomise laused andmebaasis käivitatud ning sellega loonud andmebaasis projekti tabelid ning indeksid.

- PostgreSQLi andmebaasis võite luua skeemiobjektid (nt tabelid, indeksid) skeemis *public*, kuid võite soovi korral luua ka uued skeemid. Oracle andmebaasis loote skeemiobjektid Teie kasutajanimele vastavas skeemis (C##TUD4).
- Juhul kui lausete käivitamisel ilmnes, et mõni lause ei läinud käima disaini mudelis või koodi ülevaatamisel tehtud või tegematajätud asjade pärast, siis tuleb lisaks lause parandamisele vajadusel parandada ka disaini mudelit. See, kas on mõtet genereerida mudelist kood uuesti või parandada koodi ja mudelit eraldi sõltub sellest, kui palju muudatusi Te olete eelnevalt jõudnud koodis teha. Soovitan senikaua kuni lausetes võib olla vigu panna genereeritud laused tööle ükshaaval. Niimoodi tekib Teil kõige selgem ja kiirem ülevaade, millise lause täitmine ebaõnnestus ning miks. Mõned võimalikud põhjused, miks lause käivitamine ebaõnnestub.
  - Oracle tabeli, veeru, kitsenduse või indeksi nimi on suurem kui 30 baiti (30 märki, kui kasutate märke, mille esitamiseks kulub üks bait). PostgreSQLis on makismaalne identifikaatori pikkus vaikimisi 63 baiti ning on üsna vähetõenäoline, et seda suurust ületate. Kui ületate, siis lühendab PostgreSQL automaatselt nime, mis on ka halb. Kõige tõenäolisem on see kitsenduste nimedes. Identifikaatorite suuruse kontrollimiseks võite kasutada näiteks seda vahendit: <https://mothereff.in/byte-counter>
  - Tabeli, veeru, kitsenduse või indeksi nimi sisaldab vähemalt ühte tühikut, @ märki või sidekriipsu -



Lausete käivitamiseks kasutage vahendeid (psql või PgAdmin – PostgreSQL; SQL\*Plus, SQLcl või SQL Developer – Oracle), millest oli juttu 2–4 nädala harjutustundides ja mille kohta on slaidikomplekt

**Oracle\_ja\_PostgreSQL\_kasutamine\_IDU0230\_2018.ppt**

Eelneval pildil on ekraanitõmmis PgAdmin 4 ver 3 vahendist (Te ei pea seda kasutama, psql on ka hea programm).

Looge ühendus.

Logige sisse.

Valige andmebaas.

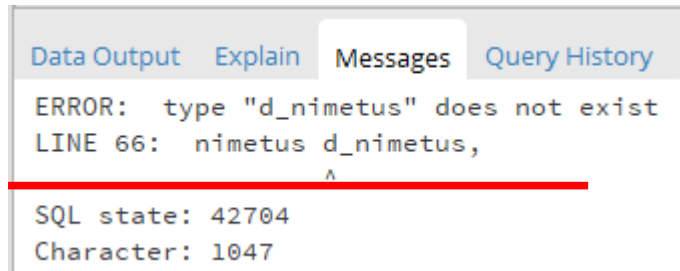
Valige *Tools => Query Tool*

Sisestage käivitav lause (või laused).

Valige *Execute/Refresh*

- ⚠ Kui teete aktiivseks ühe lause, siis käivitatakse ainult see, vastasel juhul käivitatakse järjest kõik aknas olevad laused.

Järgnev on näide veateatest, mille võite PgAdminis saada, kui käivitate *kõik laused korraga*. Veateates pole näha, millise lause/andmebaasiobjektiga seoses viga tekkis. See tuleb leida veateates esitatud rea numbri abil.



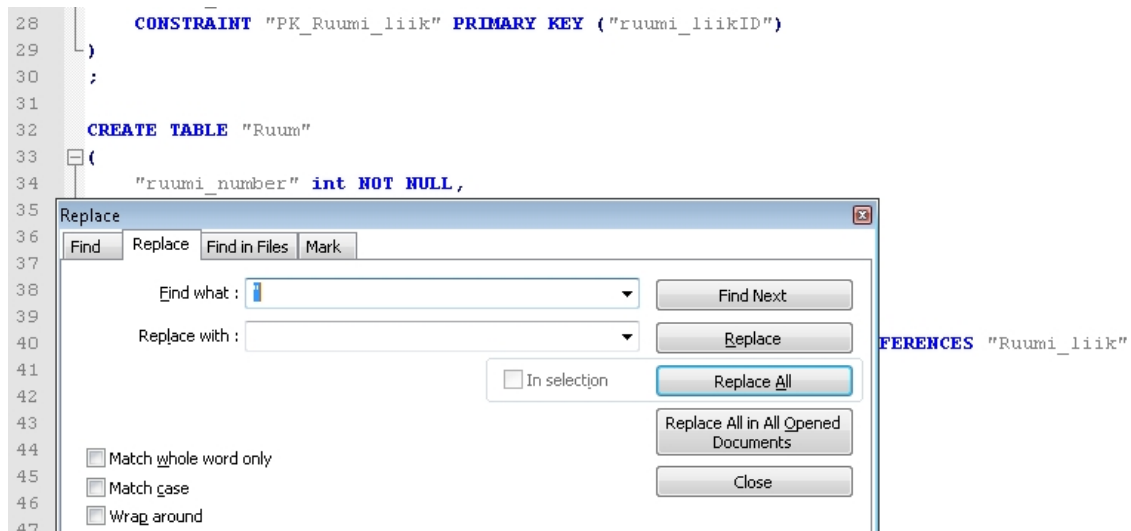
The screenshot shows the 'Messages' tab in a PostgreSQL client interface. It displays an error message: 'ERROR: type "d\_nimetus" does not exist' followed by 'LINE 66: nimetus d\_nimetus,'. A red horizontal line is drawn under the error text, and a small caret symbol (^) is positioned below the comma in the SQL line. Below the error, the 'SQL state: 42704' and 'Character: 1047' are displayed.

```
Data Output Explain Messages Query History
ERROR: type "d_nimetus" does not exist
LINE 66: nimetus d_nimetus,
                        ^
SQL state: 42704
Character: 1047
```

## Mida tuleb genereeritud lausetes ülevaadata, parandada, muuta?

- Kui genereeritud koodis on *piiritletud identifikaatorid* (jutumärkides; sellist koodi genereerib EA), siis asendage need *regulaarsete identifikaatoritega* (pole jutumärkides). Selleks tehke failis asendus (*Search and Replace*), kus asendate kõik jutumärgid mitte millegagi. See on oluline arendajate kasutusmugavuse seisukohalt:

<https://blog.codinghorror.com/the-case-for-case-insensitivity/> Regulaarsed identifikaatorid ei ole tõstutundlikud, kuid piiritletud identifikaatorid on.



<https://stackoverflow.com/questions/37111074/enterprise-architect-generate-sql-without-quotes-in-table-name/46910049#46910049> kirjutab, kuidas EA12s saada jutumärkidest lahti juba koodi genereerimisel. EA13 puhul genereeritakse nimed vaikimisi jutumärkideta.

- Kui teete projekti PostgreSQLis ning Teie teemaks on funktsionaalne allsüsteem, mis tegeleb mingi piiratud ressursi kasutamise registreerimisega, siis vaadake "Baatabelite" slaidikomplektist (lisamaterjalide lehel) *Exclusion* kitsenduse näidet – võibolla tahate seda ikkagi oma projektis kasutada. Sellisel juhul tuleb muudatusi teha koodi tasemel.

### Näited:

- Kino infosüsteemi seansside arvestus registreerib andmeid seansside registris – ühes ja samas kinosaaalis ei tohi olla selliseid seansside paare, mille toimumise aeg osaliselt või täielikult kattub.
- Hotelli infosüsteemi broneeringute arvestus registreerib andmeid broneeringute registris – ühes ja samas ruumis ei tohi olla selliseid broneeringute paare, mille periood osaliselt või täielikult kattub.
- Arvutimängude infosüsteemide meeskonnaliikmete arvestus registreerib andmeid liikmete registris – ühes ja samas meeskonnas ei tohi olla ühe ja sama isiku samas rollis liikmeks oleku paare, mille periood osaliselt või täielikult kattub.
- Treeningklubi infosüsteemi treeningute arvestus registreerib andmeid treeningute registris – ühel ja samal treeningul ei tohi sama treeneri kohta olla treeningu võimaliku läbiviimise paare, mille periood osaliselt või täielikult kattub.

- ⌘ Kui Rational Rose abil tehtud disaini mudelis on määratud veeru vaikimisi väärtuseks `CURRENT_DATE` või `CURRENT_TIMESTAMP`, siis koodigeneraator teeb vea ja jätab funktsiooni väljakutsesse kirjutamata alakriipsu.

```
CREATE TABLE Naide (
a DATE DEFAULT CURRENT DATE NOT NULL,
b TIMESTAMP WITH TIME ZONE DEFAULT CURRENT TIMESTAMP NOT
NULL) ;
```

**CURRENT DATE** tuleb asendada `CURRENT_DATE`  
**CURRENT TIMESTAMP** tuleb asendada `CURRENT_TIMESTAMP`

- ⌘ Rational Rose koodigeneraator võib (aga ei pruugi) lisada genereeritud koodi üleliigseid `DEFAULT` klausleid, kus vaikimisi väärtus pole määratud. Sellised `DEFAULT` klauslid tuleb koodist üles leida ja kustutada. Nende `DEFAULT` klauslite genereerimine on Rational Rose programmiviga.

#### Näide:

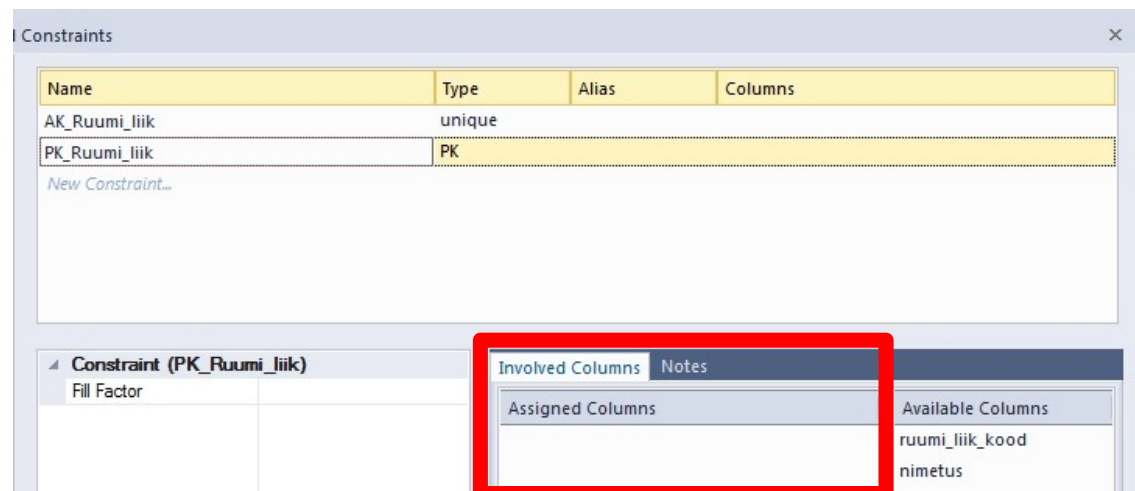
```
CREATE TABLE Koristamine (
koristamine_id INTEGER NOT NULL,
isikukood CHAR ( 11 ) DEFAULT NOT NULL,
alguse_aeg TIMESTAMP NOT NULL DEFAULT LOCALTIMESTAMP(0) ,
ruumi_number SMALLINT DEFAULT NOT NULL,
lopu_aeg TIMESTAMP NOT NULL,
kommentaar VARCHAR ( 1000 ),
CONSTRAINT PK_Koristamine PRIMARY KEY (koristamine_id),
CONSTRAINT AK_Koristamine UNIQUE (ruumi_number, isikukood,
alguse_aeg),
CONSTRAINT
chk_Koristamine_lopu_aeg_on_suurem_vordne_alguse_aeg CHECK
(lopu_aeg>=alguse_aeg),
CONSTRAINT chk_Koristamine_kommentaar_ei_koosne_tyhikutest
CHECK (kommentaar!~'^[[:space:]]*$')
) ;
```

- **Punasega DEFAULT** klauslid tuleb kustutada (kustutada sõna `DEFAULT`).
- **Sinisega DEFAULT** klausliga on kõik korras ja seda pole vaja kustutada.

- ⌘ Kui leiate koodis `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY` kitsenduse või indeksi, mille puhul pole määratud veergu/veerge, millega see on seotud, siis olete selle jätnud mudelis määramata. Parandada tuleb nii mudelit kui koodi.

#### Näide:

```
CONSTRAINT PK_Ruumi_liik PRIMARY KEY (),
```



Sidumaks kitsendusega veerge, tuleb alajaotuses *Available Columns* teha soovitud veergude nimedel topeltklõps.

- ⌘ Kui leiate koodist CHECK kitsenduse, mille avaldis pole määratud, siis olete kas jätnud mudelist CHECK kitsenduse kustutamata või olete unustanud selle avaldise kirja panna. Mõlemal juhul tuleb disaini mudelit ja koodi parandada. Kui sellise koodifragmendi genereerib EA ning mudelis on avaldis kirjeldatud, siis proovige genereerimist mõnes teises arvutis – viga võib olla konkreetsetes arvutis olevas koodigeneraatori mallis.

#### Näide:

```
CONSTRAINT chk_Ruumi_liik CHECK ( ),
```

- ⌘ Kui leiate koodist CHECK kitsenduse, mis kontrollib parooli tugevust (nt minimaalne märkide arv), siis see on viga. Tabelis parool on parooli ja soola põhjal moodustatud räsiväärtus, mis on alati ühesuguse pikkusega. Parooli tugevust tuleb kontrollida enne selle registreerimist. Kui tabelis on räsiväärtus, siis ei saa parooli kontrolliks kasutada CHECK kitsendust. See kitsendus tuleb kustutada nii mudelist kui koodist.

#### Näide:

```
CONSTRAINT chk_parool CHECK(char_length(parool)>5),
```

Parooli veerus peab olema parooli räsiväärtus, mis on leitud kasutades soola. Räsifunktsioon tagastab sõltumata sisendiks oleva stringi pikkusest alati ühesuguse pikkusega stringi. Sõltuvalt räsifunktsioonist on see pikkus erinev. Kui soovite, siis võite kasutada järgnevat kitsendust, kus n on Teie kasutatava räsifunktsiooni poolt tagastatava räsiväärtuse pikkus.

```
CONSTRAINT chk_parool CHECK(char_length(parool)=n),
```

Kui lisate sellise kitsenduse, siis muutuvad üleliigseks parooli veeruga seotud võimalikud kitsendused nagu "parool ei tohi olla tühi string või tühikutest koosnev string".

- ⌘ Kui leiate koodist CHECK kitsenduse, kus andmetega tehakse nende tüübile mittevastavaid operatsioone, siis olete teinud kitsenduse

kirjeldamisel vea. Sellise kitsenduse deklaratsiooni sisaldava lause täitmine andmebaasisüsteemis ebaõnnestub. Tuleb parandada nii disaini mudelit kui koodi.

#### Näide:

Kehtivad reeglid, et kauba kood peab olema kuus märki pikk ning kauba hind on arv, mis ei tohi olla negatiivne.

#### Valesti deklareeritud kitsendused:

```
CREATE TABLE Kaup (kauba_kood CHAR(6) NOT NULL,
hind DECIMAL(19,4) NOT NULL,
CONSTRAINT pk_kaup PRIMARY KEY (kauba_kood),
CONSTRAINT chk_kaup_kaup_kood CHECK(kauba_kood=6),
CONSTRAINT chk_kaup_hind CHECK(hind~'^.*[[:digit:]].*$'));
```

#### Õigesti deklareeritud kitsendused:

```
CREATE TABLE Kaup (kauba_kood CHAR(6) NOT NULL,
hind DECIMAL(19,4) NOT NULL,
CONSTRAINT pk_kaup PRIMARY KEY (kauba_kood),
CONSTRAINT chk_kaup_kaup_kood
CHECK(char_length(kauba_kood)=6),
CONSTRAINT chk_kaup_hind CHECK(hind>=0));
```

*char\_length* on PostgreSQL süsteemi-defineeritud funktsioon, mis leiab etteantud stringi pikkuse. *hind* on püsikoma tüüpi. Sellesse tüüpi kuuluvate väärtuste esitusviis sisaldab märki, numbraid, punkti – kõik see on ära määratud tüübiga (seda pole vaja regulaaravaldisega kontrollida). Kitsendus kontrollib, et hind ei oleks negatiivne.

- ⌘ Kui leitate koodist CHECK kitsenduse, mille avaldis ei viita tabeli veerule, või viitab tabeli veerule vale nime kasutades (suur- ja väiketähtede erinevus ei lähe arvesse, kuna kasutate regulaarseid identifikaatoreid), siis olete teinud vea disaini mudelis CHECK kitsenduse kirjeldamisel. Teil tuleb disaini mudelit ja koodi parandada.

#### Näide:

```
CONSTRAINT chk_Koristamine_kommentaar_ei_koosne_tyhikutest
CHECK ('^[[:space:]]*$')
```

```
CONSTRAINT
chk_Koristamine_lopu_aeg_on_suurem_vordne_alguse_aeg CHECK
(lopu_aeg>=alguse_aeg),
```

- ⌘ Surrogaatvõtmete korral tuleb määrata, et nende väärtused genereeritakse automaatselt. Kui tegite ülesandes 2 tehtud juhendi alusel, siis tuleb võtmeväärtuste genereerimine võtta kasutusele selliste veergude korral, mis rahuldavad **kõiki järgmiseid tingimusi**.
  - Tegemist on primaarvõtme veeruga.
  - Primaarvõti on lihtvõti.
  - Primaarvõti ei kattu (osaliselt või täielikult) välisvõtmega.
  - Primaarvõti on surrogaatvõti, mitte sisulise tähendusega võti.



- Primaarvõtme veeru nime lõpus on järelliide **\_id**.

Välisvõtme veerud ei ole kunagi SERIAL/SMALLSERIAL/BIGSERIAL (PostgreSQL) või identiteedi veerud (Oracle).

PostgreSQL korral saab järgneva muudatuse teha juba andmebaasi füüsilise disaini mudeli tasemel (vt ülesande algus).

Näide:

```
CREATE TABLE Koristamine (
  koristamine_id SERIAL NOT NULL,
```

Oracles tuleb järgnev muudatus teha genereeritud koodis, sest kahjuks ei oska meie kasutatavad CASE vahendid sellist koodi ise genereerida. Oracle (alates 12c)

Näide:

```
CREATE TABLE T990997_Koristamine (
  T990999_Koristamine_ID NUMBER(10) GENERATED AS IDENTITY
  NOT NULL,
```

Vaikimisi on identiteedi veerul ALWAYS määrang, mis tähendab, et kasutaja ei saa sellesse veergu ise väärtust lisada (väärtuse genereerib alati süsteem). NB! Ka PostgreSQLis saab (Te ei pea seda tegema) alates versioonist 10 SERIAL notatsiooni asemel määrata, et veeru näol on tegemist identiteedi veeruga.

**Välisvõtme veerud**, mille abil viidatakse eelnevalt näiteks toodud primaarvõtme veergudele, on vastavalt tüüpi INTEGER (PostgreSQL) ja NUMBER(10) (Oracle).

Tuletan veelkord meelde – sisuliste võtmete väärtuseid süsteem ei genereeri või kui genereerib, siis toimub see mingi keeruka algoritmi alusel, mis tuleb arendajal ise programmeerida. Sisuliste võtmete näited:

- ⌘ isikukood,
  - ⌘ äriregistri kood,
  - ⌘ üliõpilaskood,
  - ⌘ õppeaine kood,
  - ⌘ auto registrinumber,
  - ⌘ VIN kood,
  - ⌘ **klassifikaatori kood**.
- ⌘ Veenduge, et kui vaikimisi väärtus on staatiline **tekstiline** väärtus (ei leita funktsiooniga), siis on seda esitava literaali ümber ülakomade e apostroofid. Kui vaikimis väärtus on staatiline **arvuline** väärtus, **tõeväärtus** või leitakse **funktsiooniga**, siis ei ole väärtus ülekomade vahel.

Näide:

```
sugu CHAR ( 1 ) DEFAULT 'M' NOT NULL,
pikkus INTEGER DEFAULT 200 NOT NULL,
loomise_aeg TIMESTAMP DEFAULT LOCALTIMESTAMP(0) NOT NULL,
on_avatud BOOLEAN DEFAULT TRUE NOT NULL
```

- ⤴ PostgreSQL hoiab alles ridade muutmiseelseid versioone, et rea muutmine ei blokeeriks selle samaaegset lugemist ja vastupidi (multiversion-konkurentsjuhtimine). UPDATE lause tulemusena tekib reast uus versioon, aga ka vana versioon jääb alles ning märgitakse andmebaasis kustutatuks. Lõppkasutaja seda vana versiooni enam tabelis ei näe. Selleks, et süsteem peaks uue versiooni salvestamiseks lugema vähem plokkide, oleks hea, kui uus versioon mahuks samasse plokk, kus on vana versioon (PostgreSQL puhul kasutatakse "plokk" asemel terminit "lehekülg"). Selleks peab plokkis olema vaba ruumi. Mida vähem on operatsiooni täitmiseks andmebaasisüsteemil vaja lugeda plokkide, seda parem on see jõudlusele.

<http://blog.coelho.net/database/2014/08/23/postgresql-fillfactor-and-update.html>

Coelho (2014) katsetab selles artiklis FILLFACTOR väärtuse vähendamist ja märgib, et see suurendab plokkide ning seega salvestusmahu hulka, nõuab päringute täitmiseks suurema arvu plokkide muutmälu lugemist ja kokkuvõttes tuleks tõesti kaaluda vaid juhul, kui UPDATE operatsioone on palju ja nende jõudlus on oluline.

#### Näide:

```
CREATE TABLE Ruum (
...
kommentaar VARCHAR ( 1000 ),
...
) WITH (fillfactor=90);
/*Kuna ruumi kommentaari võidakse muuta, siis määratakse, et
tabeli plokkidesse jäetakse 10% vaba ruumi
(fillfactor=90). Vaikimisi fillfactor=100.*/
```

Kaaluge FILLFACTOR väärtuse vähendamist oma projekti põhiregistri sellistes tabelites, kus toimub UPDATE operatsioone. Operatsioonide kohta saab infot analüüsi projekti andmebaasioperatsioonide lepingutest ja CRUD maatriksist.

Mis võib vihjata sellele, et tabelis hakkab toimuma UPDATE operatsioone?

- ⤴ Tabelis on mittekohustuslikke (NULLe lubavaid) veerge, millele vastavatesse väljadesse lisatakse väärtus alles peale rea tabelisse lisamist.
- ⤴ Tabelis on välisvõtme veerg, mis viitab seisundiklassifikaatorile.
- ⤴ Tabelis on tõeväärtustüüpi veerg.
- ⤴ Tabelis on mahukaid tekstilisi väärtuseid lubav veerg.

Selliste tabelite puhul tuleks kaaluda FILLFACTOR väärtuse vähendamist.

**NB!** Oracle puhul jäetakse tabeliplokkidesse vaikimisi 10% vaba ruumi ja see võib nii jääda. Oracles reguleerib seda tabeli parameetri PCTFREE väärtus (vaikimisi väärtus on 10).

- ⤴ Kui välisvõti on *liitvõti* ja see realiseerib *mittekohustuslikku seost*, siis tuleb lisada välisvõtme deklaratsiooni MATCH FULL (PostgreSQL) või välisvõti sisaldavasse tabelisse eraldi CHECK kitsendus (Oracle), et **vältida** andmetes olukorda, mida illustreerib järgnev näide.

Näide:

Kontseptuaalne andmemudel

[Töötaja]-0..1-+lõpetab-----0..\*-[Kaup]

Tabelis *Tootaja* ei ole sellist isikukoodi. Kuna osa välisvõtme väärtusest (lop\_riik) puudub, siis süsteem lubab selliseid andmeid registreerida

Soovimatud andmed tabelis *Kaup*.*Tootaja*

isikukood	riik_kood
38808080123	EST
38808080123	LTU

*Kaup*

kaup_kood	lop_ikood	lop_riik
1	38808080123	
2	333	

Selles väljas on NULL. Kumb isik selle kauba lõpetas?

Süsteem lubab tabelis *Kaup* selliseid andmeid, sest vaikimisi on välisvõtme omadus MATCH SIMPLE. Kui välisvõtme omadus on MATCH SIMPLE ja välisvõtme väärtusest osa puudub (on NULL), siis ei pea primaartabelis sellele välisvõtme väärtusele vastet olema. MATCH SIMPLE on SQL standardi kohaselt välisvõtme *vaikimisi omadus*. Lisaks kirjeldab standard MATCH omaduse variante MATCH PARTIAL ja MATCH FULL. MATCH omaduse peale on vaja mõelda, kui tabelis on liitvälisvõti (see hõlmab rohkem kui ühe veeru), mille vähemalt ühes veerus on lubatud NULLid.

Kuidas eelnenud illustratsioonis väljatoodud andmete tabelisse *Kaup* lisamist keelata?

## PostgreSQL

```
FOREIGN KEY (lop_ikood, lop_riik) REFERENCES Tootaja
(isikukood, riik_kood) MATCH FULL ON UPDATE CASCADE
```

## Oracle.

CHECK kitsendus tabelis *Kaup*. Oracle ei võimalda FOREIGN KEY kitsenduse deklaratsioonis kasutada SQL standardis ettenähtud MATCH klauslit.

```
CHECK ((lop_ikood IS NULL AND lop_riik IS NULL) OR
(lop_ikood IS NOT NULL AND lop_riik IS NOT NULL))
```

Selliste kitsenduste korral peab igas tabeli *Kaup* reas olema registreeritud kas nii *lop\_ikood* kui ka *lop\_riik* või olema mõlemad määramata (NULLid).

- ⌘ Oracle puhul tuleb veelkord veenduda, et iga loodava tabeli, kitsenduse ja indeksi nimes sisaldub matrikli number. See on vajalik skeemis nimekonfliktide vältimiseks. Veergude nimedes ei ole matrikli numbrit vaja.

- ^ Veenduge, et tabelite, veergude, kitsenduste ja indeksite nimedes ei oleks tühikuid. SQLi regulaarsed identifikaatorid ei tohi tühikuid sisaldada. Kui seal on tühik, siis vastava lause käivitamine ebaõnnestub.
- ^ PostgreSQLis saab andmebaasiobjekti nime pikkus vaikselt olla kuni 63 baiti. Kui käivitavas lauses on see pikem, siis lühendab PostgreSQL ise nime. Kui see juhtub, tuleb vastav muudatus teha ka mudelis ning muuta nime ka lausetes, mis viitavad andmebaasiobjektile kasutades selle lühendamise-eelset nime.

```
CREATE TABLE Aken (
  aken_id SERIAL NOT NULL,
  korgus SMALLINT NOT NULL,
  CONSTRAINT pk_aken PRIMARY KEY (aken_id),
  CONSTRAINT
  chk_PostgreSQL_is_on_identifikaatori_maksimaalne_pikkus_va
  ikimisi_63_baiti_kui_pikem_siis_lyhendatakse_automaatselt
  CHECK (korgus>0));
```

```
NOTICE:  identifier
"chk_postgresql_is_on_identifikaatori_maksimaalne_pikkus_v
aikimisi_63_baiti_kui_pikem_siis_lyhendatakse_automaatselt
" will be truncated to
"chk_postgresql_is_on_identifikaatori_maksimaalne_pikkus_v
aikimi"
CREATE TABLE
```

- ^ EA11 genereerib koodi nii, et kõigepealt on CREATE TABLE laused ilma enamike kitsenduste deklaratsioonideta (v.a veergude tüübid ja NOT NULL kitsendused).

#### Näide:

```
CREATE TABLE t122164_Kirjastuse_seis_liik
(
  kirjastuse_seisundi_liik_kood  NUMBER(2) NOT NULL,
  kirjastuse_seisundi_nimetus    VARCHAR2(50) NOT NULL
)
;
```

Faili lõpus on ALTER TABLE laused, millega lisatakse tabelitele kitsendused. Tabel ja sellele rakenduvad kitsendused on loogiline tervik, kuid genereeritud failis on tabelit puudutavad laused laiali paljudes erinevates kohtades. See vähendab koodi ülevaatlikust ja vähendab koodi inspekteerimisel vigade avastamise tõenäosust.

Seega palun dokumendi jaoks laused ümber tõsta nii, et peale CREATE TABLE lauset tulevad kohe ALTER TABLE laused primaarvõtme, alternatiivvõtmete ja CHECK kitsenduste kirjeldamiseks.

**Näide:**

```
CREATE TABLE t122164_Kirjastuse_seis_liik
(
  kirjastuse_seisundi_liik_kood  NUMBER(2) NOT NULL,
  kirjastuse_seisundi_nimetus    VARCHAR2(50) NOT NULL
);

ALTER TABLE t122164_Kirjastuse_seis_liik
ADD CONSTRAINT T122164_UQ_KIRJASTUS_S_NIMI UNIQUE
(kirjastuse_seisundi_nimetus);

ALTER TABLE t122164_Kirjastuse_seis_liik
ADD CONSTRAINT t122164_CHK_kirj_S_nimi_tyhi CHECK (NOT
REGEXP_LIKE(kirjastuse_seisundi_nimetus, '^[:space:]*$')
AND NOT
REGEXP_LIKE(kirjastuse_seisundi_nimetus, '^.*[[:digit:]].*$'
));

ALTER TABLE t122164_Kirjastuse_seis_liik ADD CONSTRAINT
t122164_PK_Kirjastuse_seisund
PRIMARY KEY (kirjastuse_seisundi_liik_kood);
```

EA12 seda probleemi ei ole, kui olete valinud enne SQL lausete genereerimist *Separate Constraints from Tables = False* Seega kõige parem oleks üldse kasutada koodi genereerimiseks EA12.

- ▲ Ülesannetes 2 ja 3 nimetati indekseid (nt osaline unikaalne indeks ülesandes 2), mida ei saanud EA ja RR vahendites kirjeldada. Nende indeksite loomise kood oleks vaja nüüd kirja panna.

**Näide:** PostgreSQLis jõustatakse kitsendus, et aktiivsete õppeainete nimetused peavad olema unikaalsed.

```
CREATE TABLE Aine (aine_id SERIAL,
nimetus VARCHAR(100) NOT NULL,
on_aktiivne BOOLEAN NOT NULL DEFAULT TRUE,
CONSTRAINT pk_aine PRIMARY KEY(aine_id));

CREATE UNIQUE INDEX ak_aine_nimetus_aktiivne ON Aine
(nimetus) WHERE on_aktiivne=TRUE;

INSERT INTO Aine (nimetus) VALUES ('Andmebaasid II');
/*Lisamine õnnestub.*/

INSERT INTO Aine (nimetus) VALUES ('Andmebaasid II');
/*Lisamine ebaõnnestub.*/

INSERT INTO Aine (nimetus, on_aktiivne) VALUES
('Andmebaasid II', FALSE);
/*Lisamine õnnestub.*/
```

- EA võib Teile PostgreSQL korral genereerida sellise koodi:

```
CREATE TABLE Tootaja
(
  tootaja_id serial NOT NULL DEFAULT
nextval(('tootaja_tootaja_id_seq'::text)::regclass),

  /* Create Table Comments, Sequences for Autonumber Columns */

CREATE SEQUENCE tootaja_tootaja_id_seq INCREMENT 1 START 1;
```

Kuna SERIAL notatsiooni kasutamine tähendab, et andmebaasisüsteem loob ise arvujada generaatori ning seob selle veeruga, **siis tuleb punasega tähistatud osa kustutada**.

- Kui genereerite koodi kasutades EA12 ning määrate, et kitsenduste deklaratsioonid peavad sisalduma CREATE TABLE lauses, siis võib olla vaja muuta lausete järjekorda failis. Põhjus on selles, et kui tabeli T1 loomise lauses deklareeritakse välisvõti, mis viitab tabelile T2, siis T1 loomise lause ebaõnnestub, kui T2 on loomata. Kahjuks ei oska EA ise seda järjekorda täpselt määrata, kuid enne genereerimise algust saab arendaja ise järjekorda muuta ning muudatuse salvestada.

### Vale järjekorra näide

```
CREATE TABLE Koristamine
( ...
  CONSTRAINT FK_Koristamine_Koristaja FOREIGN KEY
  (koristaja_id) REFERENCES Koristaja (koristaja_id) ON
  DELETE No Action ON UPDATE No Action);

CREATE TABLE Koristaja
(koristaja_id integer NOT NULL,
...
  CONSTRAINT PK_Koristaja PRIMARY KEY (koristaja_id));
```

### Õige järjekord

```
CREATE TABLE Koristaja
(koristaja_id integer NOT NULL,
...
  CONSTRAINT PK_Koristaja PRIMARY KEY (koristaja_id));

CREATE TABLE Koristamine
( ...
  CONSTRAINT FK_Koristamine_Koristaja FOREIGN KEY
  (koristaja_id) REFERENCES Koristaja (koristaja_id) ON
  DELETE No Action ON UPDATE No Action);
```

Leian, et tabelite loomise koodi parim esitusviis on järgmine. Kõigepealt võiksid olla esitatud CREATE TABLE laused, mis sisaldavad kõikide ainult loodavast

tabelist sõltuvate kitsenduste deklaratsioone. Seejärel tulevad ALTER TABLE laused välisvõtmete deklareerimiseks. Samal viisil peaksid olema ka eraldi ALTER TABLE laused teiste mitut tabelit hõlmavate kitsenduste deklareerimiseks. Kuna SQL ei luba mitut tabelit hõlmavaid võtmekitsendusi ja PostgreSQL ning Oracle ei luba CHECK kitsendustes alampäringuid siis selliseid lauseid tegelikult rohkem ei ole.

Muide, sellist koodi genereerib RR. Sellise esituse korral saab nii CREATE TABLE lauseid kui ka ALTER TABLE lauseid käivitada nende lausete grupi sees suvalises järjekorras.

Selline lausete esitus oleks näide otstarbe lahususe (*separation of concerns*) disainiprintsiibi rakendamisest, sest tabelite loomisel ei pea enam arvestama CREATE TABLE lausete käivitamise järjekorraga. Selline lausete esitus vähendab nende omavahelisi sõltuvusi ning parandab kokkuvõttes koodi hallatavust ja arusaadavust.

Eelnevalt välja toodud EA ja RR vead ning puudujäägid sunnivad Teid tegema lisatööd ja on näited juhuslikust keerukusest (*accidental complexity*) ([https://en.wikipedia.org/wiki/No\\_Silver\\_Bullet](https://en.wikipedia.org/wiki/No_Silver_Bullet)). See on keerukus, mille insenerid on endale või teistele ise kaela toonud ja peavad ka ise sellega hakkama saama. Samas kulutavad nad selleks vahendeid, mida oleks ju parem kasutada süsteemi põhilise keerukusega (*essential complexity*) toimetulekuks.

Võib öelda, et programm (kui tegutseja, mis toimetab mingite algoritmide alusel) on oskuste arvutiseeritud esitus, mis peaks oma töö käigus suutma võimalikult palju olemasolevaid teadmisi ära kasutada (Arukas programm võiks veel isegi teadmisi juurde luua ning uusi oskuseid omandada). Nimetatud CASE vahendite andmete modelleerimise osa on näide olukorrast, kus teadmisi ja oskuseid juba on omajagu, kuid nende esitus tarkvaras ei ole piisavalt tasemel.

Selle ülesande lahendamiseks saab lisainfot slaidikomplektidest:

- ▲ **Baastabelid\_systeemikataloog\_IDU0230\_2018.ppt**
- ▲ **Arvujada\_generaatorid\_IDU0230\_2018.ppt**
- ▲ **Skeemid\_IDU0230\_2018.ppt**

, mille leiab <http://193.40.244.90/346>