Objekt-relatsioonilised andmebaasisüsteemid

Teema 10

Objekt-relatsioonilised (OR) andmebaasisüsteemid

- *Relatsioonilise* mudeli ristamine *objektorienteerituse (OO)* põhimõtetega.
- Mitmeid erinevaid käsitlusi selle kohta, mida objekt-relatsioonilisus tegelikult tähendab.
- Alates SQL:1999 on SQL-standardi näol tegemist *objekt-relatsioonilise* keele kirjeldusega.

OR andmebaasisüsteemide põhiideed

- Peab olema võimalus defineerida uusi *tüüpe*, kasutades muuhulgas ka *pärimisseoseid*.
- Samuti peab saama defineerida operaatoreid (funktsioone), et viia läbi operatsioone vastavat tüüpi andmetega.
- *Deklaratiivne* andmekäitluskeel peab säilima.

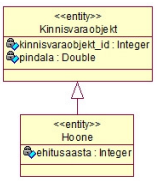
Käsitluste näiteid

- Third Generation Database System Manifesto (aastast 1990)
  - Kasutaja-defineeritud tüübid
  - Pärimisseosed tüüpide vahel
  - Kasutaja-defineeritud rutiinid
  - Reeglid, mis kirjeldavad reaktsiooni sündmustele
- The Third Manifesto (Kolmas Manifest) (esimene versioon aastast 1998)
  - Sellel põhineb ka "Andmebaasid I" tuttav käsitlus relatsioonilisest mudelist
- SQL:1999 ja hilisemad SQL standardi versioonid

Pärimine

- Üks OO lähenemise tugisambaid – saan luua uusi klasse olemasolevate põhjal
- Põhimõtteliselt võimalikud erinevad lähenemised pärimisele
  - Pärimine laiendamise teel (*specialization by extension*)
  - Pärimine kitsenduste teel (*specialization by constraint*)

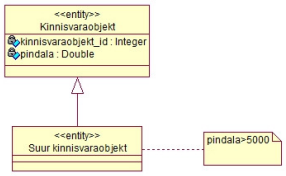
Pärimine laiendamise teel



Hoone eksemplari *ei saa* esitada ainult kinnisvaraobjekti atribuutide väärtustamise ja seoste loomise kaudu.

- Iga hoone on kinnisvaraobjekt
- Mõned kinnisvaraobjektid pole hooned
- Alamtüüp (Hoone) lisab uusi attribuute või seosetüüpe

Pärimine kitsenduste teel



- Iga suur kinnisvaraobjekt on kinnisvaraobjekt
- Mõned kinnisvaraobjektid pole suured
- Selle, kas eksemplar kuulub alamtüüpi või mitte määrab see, kas selle korral on täidetud ülatüübi atribuudi põhjal defineeritud kitsendus või mitte.

Suure kinnisvaraobjekti eksemplari *saab* esitada ainult kinnisvaraobjekti atribuutide väärtustamise ja seoste loomise kaudu.

Pärimine laiendamise *vs.* kitsenduste teel

- Objektorienteeritud võimalustega täiustatud SQLis kasutatakse laiendamise teel pärimist.
  - Tüüpide loomine pärimise kaudu
  - Tüübitud tabelite loomine pärimise kaudu
  - Tüüpimata tabelite loomine pärimise kaudu (PostgreSQL)
- SQLis ei saa kasutada pärimist kitsenduste teel.

OO võimalustega täiustatud SQLi nägemus

- *klass* = *tabel* ning *objekt* = *rida*.
  - Võimalik on tabelite loomine pärimise kaudu.
- Kas see on ikkagi õige vastavus?
  - SQLi tabelite ridade andmed ei ole meetodite taha peidetud – saan nende poole pöörduda otse.
  - Mis klassi kuuluks nt tabelite *Töötaja* ja *Osakond* kokkuühendamise tulemusena tekkinud tabelis olev rida (objekt)?

Võrdluseks – Kolmanda Manifesti nägemus

- *klass* = *andmetüüp*
- *objekt* = *tüüpi kuuluv väärtus*
- Lisaks (pole manifesti osa) esitatakse andmetüüpide pärimise kaudu loomise mudel
  - Toetab pärimist kitsenduste kaudu
  - Ei toeta pärimist laiendamise kaudu

SQL standardi areng – olulised verstapostid

- SQL:1986 – Esimene standardi versioon ANSI (*American National Standards Institute*) poolt
  - 1987 – kiidetakse heaks ka *International Organization for Standardization* (ISO) poolt
- SQL:1989
- SQL:1992 (SQL2)
- SQL:1999 (SQL3)
- SQL:2003, SQL:2006, SQL:2008 SQL:2011
- SQL:2016 – 2017. aasta sügisel kehtiv versioon

kirjeldab *objekt-relatsioonilist* andmebaasikeelt

SQL standardi areng

- SQL standardi versioon *n* = (SQL standardi versioon *n-1*) + (uus süntaks ja erisused) – (väljajäetud süntaks ja erisused)
  - *erisus = feature

SQL:1999 – standardi alamosad

- Osa 1: SQL/Framework
- Osa 2: SQL/Foundation
- Osa 3: SQL/CLI (Call-Level Interface)
- Osa 4: SQL/PSM (Persistent Stored Modules)
- Osa 5: SQL/Bindings (Host Language Bindings)

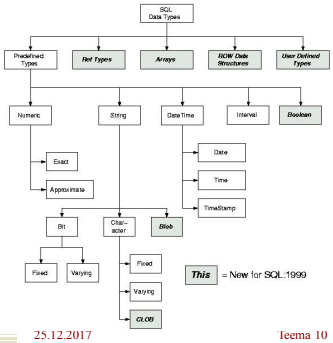
Andmebaasisüsteemi vastavus standardile (SQL:1999 ja edasi)

- Andmebaasisüsteem peab vastama *Core SQL* reeglitele.
- Lisaks Core SQLile **võivad** andmebaasisüsteemide loojad deklareerida andmebaasisüsteemi vastavust:
  - nullile või rohkemale *standardi osale*,
  - nullile või rohkemale *valikulisele erisusele* ja
  - nullile või rohkemale *paketile*
    - Enhanced datetime facilities, Enhanced integrity management, SQL/PSM & SQL/CLI, Active database ...

Tüüp

- Tüüp (andmetüüp) on nime omav lõplik väärtuste hulk.
- Kõigi hulka kuuluvate väärtustega saab teha ühesuguseid (tüübi jaoks ära määratud) operatsioone.

SQL:1999 – uued tüübid



- Uued süsteemi-defineeritud tüübid
  - -BLOB – Binary Large Object
  - -CLOB – Character Large Objects
  - -Boolean
- Tüübikonstruktorid (operaatorid tüüpide loomiseks)
- Kasutaja-defineeritud tüübid (User Defined Types – UDT)

SQL:1999 – uuendused (2)

- Tabelite loomine tüüpide põhjal
- Pärimisseosed
  - Tüüpide vahel
  - Tüübi põhjal loodud (tüübitud) tabelite vahel
- UDT tüüpi kuuluvate väärtuste järjestusreeglistik – CREATE ORDERING lause
- Kasutaja-defineeritud tüübiteisendus – CREATE CAST lause

Tüübid SQL järgi

- Baastüübid ("lihtsad" tüübid, süsteemi-defineeritud skalaarsed tüübid)
- Kasutaja-defineeritud tüübid (*user-defined types* – UDT) (alates SQL:1999):
  - baastüübil põhinev – *distinct type*,
    - SQL:2016 lubab luua ka kollektsioonitüübi põhjal.
  - struktureeritud tüüp – *structured type*.
- Tüübid, mis on loodud *tüübikonstruktori* abil (alates SQL:1999)

SQLi kasutaja-defineeritud tüübid

- Tüübi loomisel **ei saa** kasutada *deklaratiivseid kitsendusi* (tüübi kitsendusi), et piirata tüüpi kuuluvate väärtuste hulka.
- Tüübi loomisel pärimise kaudu **ei saa** kasutada deklaratiivseid kitsendusi (*specialization by constraint*), et piirata alamtüüpi kuuluvate väärtuste hulka.
- Tüübil on *meetodid* (funktsioonid), mille abil saab vastavat tüüpi väärtust kasutada.
- Võimaldavad jõustada *tugeva tüüpimise*.

Tüüp *vs*. domeen SQLis

- Tüüp ja domeen on SQLis kaks ise asja.
  - Domeen on SQLis nime omav kasutaja-defineeritud objekt, mille abil saab määrata baastabelite veergude omadusi.
  - Domeeni kasutatakse *alternatiivina* tüübile.
  - Domeeni baastüübiks ei saa olla kasutaja-defineeritud tüüp.

Distinct type

- CREATE TYPE aine_kood_t AS CHAR(7) FINAL;
- CREATE TYPE matrili_nr_t_kood AS CHAR(6) FINAL;
- CREATE TABLE Aine (... aine_kood aine_kood_t ...);
- SELECT * FROM Aine WHERE aine_kood=CAST('IDU3381' AS aine_kood_t);
  - CAST('IDU3381' AS aine_kood_t) – tüübiteisenduse rakendamine stringitüüpi literaalile

Distinct type (2)

- Võib põhineda *süsteemi-defineeritud tüübil* või *kollektsioonitüübil* (loodud ARRAY või MULTISET tüübikonstruktori abil).
- Ei saa olla *alamtüüpe.*
- Luuakse automaatselt *tüübiteisenduse funktsioonid* ja tüübiteisendus.
  - *aine_kood_t* tüüpi väärtus => *CHAR* tüüpi väärtus
  - *CHAR* tüüpi väärtus => *aine_kood_t* tüüpi väärtus

Tugev tüüpimine

- 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.
  - Operatsiooni läbiviimiseks peab leiduma sobivat tüüpi parameetritega operaator.

Tugev tüüpimine – tualeti näide

- Naised ja mehed kui tüübid.
  - Nime omav, lõplik isikute hulk.
- Tualettruumi külastus kui operatsioon.
- Vastassoo tualettruumi külastus.
  - see võib toimuda mingi kehtestatud korra kohaselt (nagu operaator);
  - inimene võib teha soovahetuse (tüübiteisendus);
  - inimene loobub külastusest.

Tugev tüüpimine – andmebaasi näide

- Väärtusega saab teha vaid neid operatsioone, mida andmebaasi programmeerija on väärtuse tüübi jaoks ette näinud.
  - Kui keegi soovib aine koodile rakendada stringifunktsioone, tuleb eelnevalt läbi viia tüübiteisendus (ingl *casting*).
  - Kui keegi soovib võrrelda aine koode ja matrikli numbreid, tuleb läbi viia tüübiteisendus või kirjutada uus võrdlusfunktsioon.
  - Muutujale, mis on tüüpi *aine_kood_t*, ei saa omistada väärtust, mis on tüüpi *matrikli_nr_t* ja vastupidi.
  - Väärtust, mis on tüüpi *aine_kood_t*, ei saa lisada veergu, mis on tüüpi *matrikli_nr_t* ja vastupidi.

Struktureeritud tüüp

- Kirjeldatakse *atribuudid* ja *meetodid*.
- Võib luua olemasolevast struktureeritud tüübist *pärimise* kaudu.
- Igal atribuudil on *tüüp*.
- Automaatselt luuakse *konstruktori meetod* – funktsioon vastavat tüüpi objekti loomiseks
- Meetod on tegelikult tüübiga seotud *funktsioon*.

Struktureeritud tüüp (2)

- Meetodid on kirjutatud mingis imperatiivses keeles.
  - Imperatiivses keeles kirjeldatakse tegevuse algoritm (vastandina oodatava tulemuse kirjeldamisele deklaratiivses keeles)
- Kapseldamine:
  - atribuudi väärtuseid saab lugeda/muuta meetodite kaudu,
  - atribuudi defineerimisel luuakse automaatselt kaks meetodit – väärtuse lugemiseks ja muutmiseks.
- Kasutaja-defineeritud struktureeritud tüüpi võib kasutada veeru tüübina või luua selle põhjal tabeli.

Struktureeritud tüüp – näide

```
CREATE TYPE Aine_tyyp AS
aine_kood CHAR(7),
nimetus VARCHAR(200),
ainepunkte DECIMAL(3,1), --Kitsendust ei saa luua
loomise_aeg DATE,
kinnitamise_aeg DATE,
NOT FINAL --Tüübile saab luua alamtüüpe
METHOD vanus() RETURNS INTEGER;
--Deklareerin, et eksisteerib funktsioon vanus()
```


Automaatselt loodud meetodid

- Konstruktor
- Iga atribuudi jaoks:
  - vaatleja,
  - muutja.

Struktureeritud tüüp – meetodi spetsifikatsioon

```
CREATE METHOD vanus() RETURNS
  INTEGER
FOR Aine_tyyp
BEGIN
  RETURN Now()-loomise_aeg;
END;
```

Veerg, mis on UDT tüüpi

- CREATE TABLE Aine(aine_id INTEGER PRIMARY KEY, aine_veerg Aine_tyyp);
- Päringu tegemise näide:
  - SELECT aine_veerg.aine_kood(), aine_veerg.nimetus() FROM Aine;
- aine_kood() – pöördumine vaatleja meetodi poole
- "Tavalise" tabeli saab kasutajale esitada vaatena

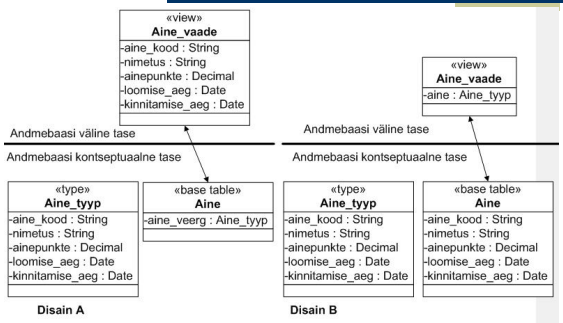
Veerg, mis on UDT tüüpi – rea lisamine

```
BEGIN
   DECLARE a Aine_tyyp; --ajutise muutuja deklareerimine
   SET a=Aine_tyyp(); --konstruktori poole pöördumine
   SET a=a.aine_kood('IDU0230'); --muutja poole pöördumine
   SET a=a.nimetus('Andmebaasid II');
   SET a=a.ainepunkte(3.5);
   SET a=a.loomise_aeg('2009-01-09');
   SET a=a.kinnitamise_aeg('2009-01-13');
   INSERT INTO Aine(aine_id, aine_veerg) VALUES(1, a);
END;
```


Tabel, mis on UDT tüüpi

- Lähtub eeldustest
  - Tabel = Objektitüüp
  - Tabeli rida = Objekti eksemplar (Objekt)
- CREATE TABLE Aine OF **Aine_tyyp** (REF IS AineID SYSTEM GENERATED);

Disaini alternatiivid



Disaini alternatiivid – soovitusi

- Disaini B korral on andmebaasi tasemel lihtsam teha päringuid ja jõustada deklaratiivselt kitsendusi.
- Ärge looge tüübi (*Aine_tyyp*) põhjal tabelit (tüübitud tabelit).
  - Tabeli loomise ja kasutamise keerukus kasvab, kuid saadav kasu on väga küsitav.

Tüübikonstruktorid

- Täiendav võimalus defineerida uusi tüüpe.
- Tüübikonstruktor on spetsiifiline *operaator*, mis tagastab tüübi.

Tüübikonstruktorid (2)

- **ROW** – reatüübi konstruktor (alates SQL:1999).
  - Reatüüp on väljanimede ja tüüpide paaride jada
  - *Näide*: ROW (maakond VARCHAR(30), linn VARCHAR(50), tänav VARCHAR(50))
- **REF** – viitetüübi konstruktor (alates SQL:1999).
  - Struktureeritud tüübi ST loomisel luuakse ka automaatselt viitetüüp REF(ST)

Tüübikonstruktorid (3)

- *Kollektsioon* – liitväärtus, mis koosneb nullist või rohkemast ühte tüüpi elemendist.
  - Elementide tüübid võivad ka olla erinevad, kuid kõik tüübid peavad kuuluma ühte tüüpide hierarhiasse.
- Kollektsioonitüübi konstruktor: **ARRAY** – massiivitüübi konstruktor (alates SQL:1999).
  - Massiivis on ühte tüüpi elemendid.
  - Massiivis võib olla korduvaid elemente.
  - Igal massiivi kuuluval elemendil on massiivis positsioon (elemendid on *järjestatud*).
  - Tüübi konstrueerimisel võib määrata maksimaalse elementide arvu.

Massiivitüüpide näiteid

- Tüübikonstruktori abil saab konstrueerida suure hulga erinevaid tüüpe.
  - INTEGER ARRAY [5]
    - Täisarvude massiiv, kuni 5 elementi
    - Väärtuse näide: (1, 23, 4, 98, 102)
  - VARCHAR(50) ARRAY [10]
  - VARCHAR(55) ARRAY
    - Maksimaalne elementide arv pole määratud
  - matrikli_nr_t ARRAY [12]
  - DECIMAL(10,2) ARRAY [5]
  - ...

Tüübikonstruktorid (4)

- Kollektsioonitüübi konstruktor: **MULTISET** (alates SQL:2003)
  - Multiset e multihulk on ühte tüüpi elementide *järjestamata* hulk.
  - Üks element võib hulgas esineda *korduvalt*.
  - Tüübi konstrueerimisel ei saa määrata maksimaalset elementide hulka.
  - Veerg, mis on ridade multihulga tüüpi – piltlikult öeldes on veerule vastavates väljades tabelid (ingl *nested table*).

Konstrueeritud tüübi kasutamine – näited

Isik_1

isikukood	perenimi	kodune_aadress
30505661234	Kuslap	ROW('Ehitajate tee 5','Tallinn','19086')

Reatüübi kasutamine

Isik_2

isikukood	perenimi	e_mail
30505661234	Kuslap	(kuslap@hot.ee; kuslap@mail.ee; kuslap@hotmail.com)

Massiivitüübi kasutamine

Isik_4

isikukood	perenimi	hobid
30505661234	Kuslap	ROW('suusatamine',2000) ROW('ujumine',1990) ROW('sukeldumine',2005)

Multihulga tüübi kasutamine

Kuidas registreerida hobisid, mis pole seotud ühegi isikuga?

- Luua tabel:
  Hobi (nimetus) Primaarvõti (nimetus)
  - Keerukus kasvab.
    - Tabelisse *Isik_4* uue rea lisamisel või väärtuse muutmisel veerus *hobid* tuleb eraldi kontrollida, et veerule *hobid* vastavas väljas olevas uues liitväärtuses sisalduv väärtus oleks ka registreeritud tabelis *Hobi*.
    - Tuleb tagada, et tabelist *Hobi* rea kustutamine ebaõnnestuks, kui see hobi on mõne isiku hobiks?

Disaini soovitus

- Vältida ridade multihulga tüüpi veergude kasutamist baastabelites ja kui vaja, siis kasutada neid vaadetes.

Konstrueeritud tüübi kasutamine – näited (2)

Isik_3

isikukood	perenimi	õppimised
30505661234	Kuslap	

Oppimine

aine_kood	tulemus	eksami_kuupäev
IDU3381	4	2005-01-04
IDU3382	5	2005-01-09

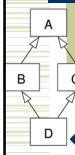
Viitade kollektsioon

Päringute süntaks muutub:

SELECT I.isikukood FROM Isik_3 I WHERE **I.õppimised.aine_kood='IDU3381'**;

Konstrueeritud tüübi kasutamine – näited (3)

- CREATE TYPE Aine_tyyp AS ... --5 atribuuti
- CREATE TABLE Aine OF Aine_tyyp (**REF IS AineID SYSTEM GENERATED**); --6 veergu
  - Tüübitud tabel (ingl *typed table*)
- CREATE TABLE Aine_kirjeldus (aine_kirjeldus_id INTEGER PRIMARY KEY, tekst VARCHAR(1000), loomise_aeg DATE, **aine *REF(Aine_tyyp)* SCOPE Aine REFERENCES ARE CHECKED ON DELETE CASCADE**);
  - Veerus *aine* olev väärtus on viide tüübitud tabeli *Aine* reale.



Pärimine

- Atribuudid ja meetodid kanduvad tuletatud tüüpi.
- Pärimine kasutaja-defineeritud tüüpide vahel:
  - CREATE TYPE Isik ...;
  - CREATE TYPE Yliopilane ... UNDER Isik;
- Pärimine kasutaja-defineeritud tüübi põhjal loodud tabelite (tüübitud tabelite) vahel:
  - CREATE TABLE Isikud OF Isik(...);
  - CREATE TABLE Yliopilased OF Yliopilane UNDER Isikud;
- SQL ei toeta praegu *mitmest pärimist*.

Pärimine tabelite vahel

- PostgreSQLis ei pea tabelite vahelise pärimissuhte kasutamiseks olema tegu tüübitud tabelitega.
- Igal juhul tekitab asjatut keerukust.
- Parema lahenduse saab realiseerida kasutades vaateid, trigereid ja funktsioone ning ilma pärimist kasutamata.
- PostgreSQLis saab tabelite vahelist pärimist kasutades realiseerida tabeli sektsioonideks jagamist.

Võimalike disainilahenduste arvu kasv

- Soutou, C., 2001. Modeling Relationships in Object-Relational Databases. *Data & Knowledge Engineering*, Vol. 36, No. 1, pp. 79-107.
  - Toob näite, et SQL:1992 andmebaasisüsteemis saaks 1:M seosetüüpi realiseerida kahel erineval viisil, kuid objekt-relatsioonilises (SQL:1999) andmebaasisüsteemis 12-l erineval viisil.
- Suurem võimalus teha halb valik.

SQL:1999 –uuendused (3)

- SIMILAR ja DISTINCT predikaadid.
- Lubatud on muuta andmeid baastabelites läbi vaadete, mille alampäring sisaldab tabelite ühendamist.

SQL:1999 –uuendused (4)

- Rekursiivsed päringud – päringud hierarhiliste andmete põhjal
- CREATE TABLE LIKE lause – tabeli loomine olemasoleva tabeli põhjal
- Trigerid – aktiivne andmebaas
- Rollid – õiguste jagamine
- Uut liiki kasutaja-defineeritud rutiinid – *meetodid*

SQL:1999 –uuendused (5)

- INFORMATION_SCHEMA – vaated süsteemikataloogi põhjal. Nende vaadete kaudu ei saa muuta andmeid baastabelites (süsteemikataloogis)
- Salvestuspunktid transaktsioonides
- Välised tabelid – andmebaasist saab küsida väljaspool andmebaasi asuvaid andmeid

Kasutaja-defineeritud rutiinid SQLis

- Võimalust taolisi rutiine luua loetakse mõnede autorite poolt objekt-relatsioonilise andmebaasikeele üheks tunnuseks.
- SQL standardis aastast 1996:
  - SQL:92/PSM (ka PSM-96)
- SQL keelt on laiendatud protseduursete lausekonstruktsioonidega – muutujad, CASE-, IF-, LOOP-, WHILE-, FOR- laused jne.
- Andmebaasi kasutajad saavad luua rutiine (*User Defined Routines*)

Kasutaja-defineeritud rutiinid SQLis (2)

- Rutiinide liigitus
  - Protseduurid, funktsioonid, tüüpide meetodid (viimased alates SQL:1999)
- Rutiinide liigitus keele järgi
  - SQL rutiinid – kirjutatud SQL keeles
  - Välised rutiinid – kirjutatud mõnes kõrgtaseme programmeerimiskeeles (nt C, Java)
    - Paiknevad kompileeritud kujul operatsioonisüsteemi failisüsteemi failides.
- Rutiine saab koondada moodulitesse

Kasutaja-defineeritud funktsioonide näiteid

- Kasutaja-defineeritud skalaarsed funktsioonid
  - *User Defined Scalar Functions* (UDSF)
  - Parameetrid ja tulemus skalaarset tüüpi
- Kasutaja-defineeritud grupifunktsioonid
  - *User Defined Aggregate Functions* (UDAG)
  - Parameetrid ridade multihulga tüüpi, tulemus skalaarset tüüpi.

Kasutaja-defineeritud funktsioonide näiteid (2)

- Kasutaja-defineeritud tabelifunktsioonid
  - *User Defined Table Functions* (UDTF)
  - Parameetrid skalaarset tüüpi, tulemus ridade multihulga tüüpi

SQL:2003 – standardi alamosad

- Osa 1: SQL/Framework
- Osa 2: SQL/Foundation
- Osa 3: SQL/CLI (Call-Level Interface)
- Osa 4: SQL/PSM (Persistent Stored Modules)

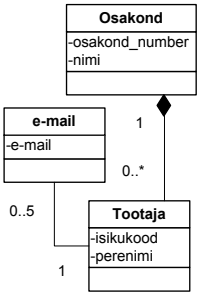
SQL:2003 – uued standardi alamosad

- Osa 9: SQL/MED (Management of External Data)
- Osa 10: SQL/OLB (Object Language Binding) (SQLJ0)
  - SQLi sissepõimimine Java programmi
- Osa 11: SQL/Schemata
  - INFORMATION_SCHEMA
- Osa 13: SQL/JRT (Java Routines and Types) (SQLJ1, SQLJ2)
- Osa 14: SQL/XML

SQL:2003 – uuendused

- SQL/XML
  - Süsteemi-defineeritud XML tüüp ja seotud operaatorid
  - Funktsioonid SQL päringus tulemuse põhjal XML tüüpi väärtuse koostamiseks
- Veel uusi tüüpe
  - Bigint
  - MULTISET tüübikonstruktor ja funktsioonid seda tüüpi väärtuste töötlemiseks
    - Multiset e multihulk – muutuva pikkusega, järjestamata kollektsioon ühetüübilisi elemente

Multihulga tüüpi veerg



- CREATE TABLE Osakond (osakond_number INTEGER PRIMARY KEY, nimi VARCHAR(50), tootaja **ROW (isikukood CHAR(11), perenimi VARCHAR(1000), e_mail *VARCHAR(254) ARRAY [5]*) MULTISET**);

Multihulk – päringud

- Leian osakonnas nr 20 töötavate töötajate arvu (kordused on eemaldatud):
  - SELECT **CARDINALITY(SET(tootaja)**) AS card FROM Osakond WHERE osakond_number=20;
- Leian osakondade nimede hulga (kordused on eemaldatud):
  - SELECT **SET**(**COLLECT**(nimi)) AS osakondade_nimed FROM Osakond;

Multihulk – päringud (2)

- SELECT SUM (t.c) summa FROM **UNNEST** (MULTISET [2, 3, 5, 7]) AS t(c);
- summa=17

SQL:2003 – uuendused (2)

- Võimalus tabeli loomisel deklareerida, et tabeli veergu tuleb väärtus automaatselt arvutada – genereeritav veerg.
- CREATE TABLE AS lause – tabeli loomine päringu tulemuse põhjal.
- MERGE lause (INSERT ja UPDATE lause kombinatsioon).
- Lause, mis leiab päringu tulemuse juhuslikult valitud ridade puhul (kasulik nt järeldusrünnete tõrjumiseks).

SQL:2003 – uuendused (3)

- Surrogaatvõtme väärtustamine:
  - arvujada generaatori objekt,
  - *identity* veerg.
- Tabelifunktsioonid, mis tagastavad ridade multihulga.
- SQL *Invoked* Routines – rutiini täitmisel kasutatakse väljakutsuja, mitte rutiini looja õiguseid.
- Aknafunktsioonid (ingl *window functions*) – RANK(), DENSE_RANK(), ROW_NUMBER(), ...

SQL:2006, SQL:2008 – uuendused

- SQL:2006
  - Täiendavad võimalused XML formaadis andmete haldamiseks SQL-andmebaasis
- SQL:2008
  - TRUNCATE TABLE lause
  - INSTEAD OF trigerid
  - ALTER COLUMN SET DATA TYPE lause
  - …

SQL:2011 – uuendused

- Näeb ette spetsiaalsed vahendid andmebaasis ajaandmete hoidmise võimaldamiseks (sealhulgas ajaloolised andmed).
- On võimalik andmebaasis deklareeritud kitsendusi sisse- ja välja lülitada.
- SELECT lausete sees saab anda korralduse andmemuudatuse läbiviimiseks (täita INSERT, UPDATE, DELETE või MERGE lause).
- …

SQL: 2016 – uuendused

- Ridade gruppide leidmine, kus järjestikulised read moodustavad mingi mustri.
  - *match_recognize* klausel (Olemas Oracle 12c Release 1)
- JSON'i tugi.
  - Ei nähta ette eraldi JSON tüüpi, kuid nähake ette hulk funktsioone ja operaatoreid
- *Listagg* kokkuvõttefunktsioon.
  - Ühendab gruppi kuuluvad stringid kokku üheks stringiks
- …

Objekt-relatsiooniliste SQL-andmebaasisüsteemide näiteid

- PostgreSQL
- Oracle (alates 8i)
- IBM DB2

PostgreSQL (10) kui ORDBMS

- Süsteemi-defineeritud **geomeetrilised** ja **internetiaadresside** esitamiseks mõeldud **tüübid** ning seotud funktsioonid ja operaatorid
- Süsteemi-defineeritud **vahemiku tüübid** ning seotud funktsioonid ja operaatorid
  - Iga sellise tüübi korral esitavad sellesse kuuluvad väärtused mingit ühte kindlat tüüpi väärtuste vahemikke

PostgreSQL (10) kui ORDBMS (2)

- **Kasutaja-defineeritud tüüpide** loomise võimalus
  - saab luua baastüüpe (skalaarseid tüüpe), vahemiku tüüpe, liittüüpe, loendtüüpe
- Kasutaja-defineeritud tüüpi kuuluvate andmete kasutamiseks mõeldud **operaatorite** ja **funktsioonide** loomise võimalus
- **ARRAY tüübikonstruktor**

Geomeetrilised tüübid

- CREATE TABLE Kujund (kujund_id serial CONSTRAINT pk_kujund PRIMARY KEY, ristkylik box);
- INSERT INTO Kujund (ristkylik) VALUES ('(-1,1),(1,-1)');
- INSERT INTO Kujund (ristkylik) VALUES ('(-2,1),(1,-3)');
- INSERT INTO Kujund (ristkylik) VALUES ('(2, 3),(3, 2)');

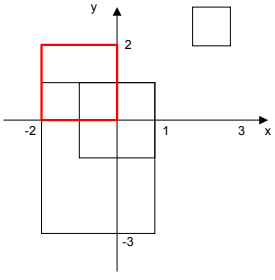
Geomeetrilised tüübid (2)

- Leian kujundi pindala, laiuse ja kõrguse:
- SELECT area(ristkylik) AS pindala, height(ristkylik) AS korgus, width(ristkylik) AS laius FROM Kujund;

Geomeetrilised tüübid (3)

- Leian etteantud koordinaatidega ristkülikuga kattuvad ristkülikud:
  - && – Overlaps operaator
- SELECT * FROM Kujund WHERE ristkylik && ('(-2,2),(0,0)');

```
kujund_id |   ristkylik
----------+---------------
         1 | (1,1),(-1,-1)
         2 | (1,1),(-2,-3)
(2 rows)
```



Veel PostgreSQLi süsteemi-defineeritud tüüpe

- *xml* – XML tüüp.
  - Lisaks pakub PostgreSQL funktsioone XML formaadis andmete kasutamiseks – nt funktsioon kontrollimaks, et dokument on trimmis.
- *uuid* – Universally Unique Identifier.
  - Tüüpi kuuluvad väärtused on 128 bitised, globaalselt unikaalsed (unikaalsus teiste sama algoritmiga genereeritud väärtuste hulgas).
  - Väärtuse näide: a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11

Veel PostgreSQLi süsteemi-defineeritud tüüpe (2)

- *JSON* ja *JSONB* – JavaScript Object Notation
  - Tüüpi kuuluvad väärtused on JSON kergekaalulises andmevahetusformaadis esitatud andmehulgad.
  - CREATE TABLE Perekond(perekond_id SERIAL PRIMARY KEY, kirjeldus **JSONB** NOT NULL);

Veel PostgreSQLi süsteemi-defineeritud tüüpe (3)

```
erki=# INSERT INTO Perekond (kirjeldus) VALUES (
erki(# '{
erki'# "nimi": "Juhan",
erki'# "nimi": "Juhan",
erki'# "vanus": 42,
erki'# "lapsed": [
erki'# {
erki'# "nimi": "Mari",
erki'# "vanus": 19
erki'# },
erki'# {
erki'# "nimi": "Toomas",
erki'# "vanus": 22
erki'# }
erki'# ]
erki'# }');
INSERT 0 1
erki=# SELECT * FROM Perekond;
 perekond_id |                                   kirjeldus
-------------+-------------------------------------------------------------------------------------------
           1 | {"nimi": "Juhan", "vanus": 42, "lapsed": [{"nimi": "Mari", "vanus": 19}, {"nimi": "Toomas", "vanus": 22}]}
(1 row)
```

Üleliigsed tühikud eemaldati, korduvad võtme-väärtuse paarid eemaldati.

Kasutaja-defineeritud vahemiku tüüp

- CREATE TYPE timerange AS RANGE (subtype = time);
- CREATE TABLE Broneering(broneering_id SERIAL PRIMARY KEY, seeria_nr INTEGER NOT NULL, periood timerange NOT NULL);
- INSERT INTO Broneering (seeria_nr, periood) VALUES(111, '(12:00, 13:00]'); -- (ja) – otspunkt pole kaasa arvatud
- INSERT INTO Broneering (seeria_nr, periood) VALUES(111, '[13:00, 11:00]'); -- [ja] – otspunkt kaasa arvatud
  - /*ERROR: range lower bound must be less than or equal to range upper bound*/

Vahemiku tüüpi veerg ja *exclusion* kitsendus

- CREATE EXTENSION IF NOT EXISTS btree_gist;
- ALTER TABLE Broneering ADD EXCLUDE USING gist (**seeria_nr WITH =**, periood WITH &&);
  - Loodud kitsendus tagab, et tabelis *Broneering* ei tohi leiduda ühtegi ridade paari {r1, r2}, mille korral järgneva avaldise kontrollimine annab tulemuseks TRUE.
  - **r1.seeria_nr=r2.seeria_nr** AND **r1.periood && r2.periood**
- INSERT INTO Broneering (seeria_nr, periood) VALUES(111, '[12:10, 13:30]');
  - /*ERROR: conflicting key value violates exclusion constraint "broneering_seeria_nr_periood_excl " */

Liittüübi ja tabeli loomine

- CREATE TYPE **aadress_t** AS (aadress VARCHAR(100), linn VARCHAR(100), postikood VARCHAR(10));
- CREATE TABLE Isik(isik_id INTEGER PRIMARY KEY, perenimi VARCHAR(1000) NOT NULL,
  /*Massiivitüüpi veerg*/
  e_mail **VARCHAR(254)[]** NOT NULL,
  kodune_aadress **aadress_t** NOT NULL ,
  CONSTRAINT chk_kodune_aadress_linn CHECK(length((kodune_aadress).postikood)>=5)));

NB! Tabeli loomisel loodi automaatselt liittüüp *Isik*, mille struktuur vastab tabelis *Isik* lubatud ridade struktuurile.

Andmete lisamine

- INSERT INTO Isik(isik_id, perenimi, e_mail, kodune_aadress) VALUES (1, 'Mets','{"t990999@ttu.ee", "metsamees@hot.ee"}', **ROW('Ehitajate tee 5','Tallinn', '19086')**);

Disaini soovitus

- Baastabelis võib kaaluda *massiivitüüpi* veeru kasutamist, et realiseerida analüüsi käigus leitud *mitmeväärtuseline* atribuut (atribuut, mis ühe olemi kontekstis võib omada mitut väärtust).
- Sellel lahendusel on ka puuduseid – näiteks tuleb korduvate massiivi elementide vältimiseks kasutada tabeliga seotud trigerit.

Andmete otsimine

- Leia Tallinnas elavate isikute isik_id, perenimi, esimene e-mail (massiivis) ja koduse aadressi postikood.
  - SELECT isik_id, perenimi, e_mail[1], (kodune_aadress).postikood FROM Isik WHERE (kodune_aadress).linn='Tallinn';
- Leia andmed isikute kohta, kelle e-mail on metsamees@hot.ee.
  - SELECT * FROM Isik WHERE 'metsamees@hot.ee' = ANY(e_mail);

Andmete otsimine (2)

- Teisenda e-meilide massiiv tabeli ridadeks. Väljasta igas reas isik_id ja perenimi.
- Väljasta igas reas järjekorranumber, mis näitab aadressi paiknemist massiivis.
  - SELECT isik_id, perenimi, e_meil, jrk FROM Isik, **unnest**(e_mail) WITH ORDINALITY e(e_meil, jrk);
  - Järjekorranumber võib tähendada *tähtsust*, kuid siis puudub võimalus väljendada *võrdtähtsust*.

Andmete otsimine (3)

- Kui on vaja registreerida iga aadressi tähtsus, siis eelista *mitmemõõtmelist massiivi*, kus ühes mõõtmes tähtsus ja teises aadress.
- Leia isikud, kes elavad kindlal aadressil.
  - SELECT * FROM Isik WHERE kodune_aadress= CAST (ROW('Ehitajate tee 5','Tallinn', '19086') AS aadress_t);
    - CAST – tüübiteisendus

Operaatori funktsiooni loomine

```
CREATE OR REPLACE FUNCTION
  aadress_t_compare_piirkond (aadress_t, aadress_t)
  RETURNS BOOLEAN AS $$
BEGIN
RETURN upper($1.linn)=upper($2.linn) AND
  upper($1.postikood)=upper($2.postikood);
END;
$$ LANGUAGE plpgsql;
```

Operaatori loomine

```
CREATE OPERATOR && (
    leftarg = aadress_t,
    rightarg = aadress_t,
    procedure = aadress_t_compare_piirkond
);
```


Operaator

- Selleks, et aadressid a1 ja a2 oleksid samas piirkonnas peab kehtima tingimus: *a1.linn=a2.linn ja a1.postikood=a2.postikood*.
- Operaatori kasutamine on tegelikult funktsiooni poole pöördumine.
- Operaatori funktsioon võib kasutada olemasolevaid operaatoreid ja funktsioone:
  - Nt upper($1.linn)=upper($2.linn) – kahe stringi võrdlemine, kusjuures võrreldavates stringides on kõik tähed teisendatud suurteks tähtedeks.

Andmete otsimine (3)

- SELECT * FROM Isik WHERE kodune_aadress **&&** CAST (ROW('Ehitajate tee 6','TALlinn', '19086') AS aadress_t);
  - Päring leiab ühe rea.
- SELECT * FROM Isik WHERE kodune_aadress **&&** CAST (ROW('Ehitajate tee 5','Tartu', '19086') AS aadress_t);
  - Päring ei leia ühtegi rida.

Skalaarne tüüp *vs*. reatüüp

- Eelnevalt näidatud CREATE TYPE lause PostgreSQLis loob sisuliselt uue reatüübi.
- SQL standard ei näe ette võimalust luua reatüüpe, et neid *hiljem* kasutada (nagu skalaarseid tüüpe).
- SQL standardi kohaselt:
  - reatüübi loomiseks pöördutakse vajalikus kohas ROW tüübikonstruktori poole,
  - reatüübile ei anta eraldi nime (nt *Isik*).

Skalaarne tüüp *vs*. reatüüp (2)

- Skalaarsete tüüpide puhul kehtib kapseldamise omadus – skalaarse tüübi atribuudi väärtuseid saab lugeda ja muuta meetodite kaudu.
- Reatüüpi väärtusel on kogum kasutajale nähtavaid (avalikke) atribuute, mille poole saab pöörduda ilma meetodeid kasutamata.

PostgreSQL – loendtüüp

- CREATE TYPE sugu_t AS ENUM ('0','1','2','9');
- CREATE TYPE iseloom_t AS ENUM ('koleerik', 'sangviinik', 'melanhoolik','flegmaatik');
- CREATE TABLE Tegelane(tegelane_id SERIAL PRIMARY KEY, perenimi VARCHAR(1000) NOT NULL, sugu sugu_t NOT NULL, iseloom iseloom_t NOT NULL);

PostgreSQL – loendtüüp (2)

- *Tugeva tüüpimise* näide – erinevatesse loendtüüpidesse kuuluvate väärtuste võrdlemiseks vajalik tüübiteisendus või uus võrdlusoperaator.
  - SELECT * FROM Tegelane WHERE sugu=iseloom;
    - ERROR: operator does not exist: sugu_t = iseloom_t
  - SELECT * FROM Tegelane WHERE CAST(sugu AS VARCHAR)=CAST(iseloom AS VARCHAR);
    - Päring täidetakse, aga mis oleks tulemuse sisuline mõte?

PostgreSQL kui ORDBMS – vähemkasulikud võimalused

- Tabeli loomisel saab määrata, kas siduda tabeli iga reaga **Object Identifier**, mida kasutaja päringu tulemuses vaikimisi ei näe
- **Pärimisseos** tüüpimata tabelite vahel
  - PostgreSQL võimaldab luua uue tabeli T' olemasoleva tabeli T põhjal kasutades INHERITS klauslit.
  - Tabelis T' samad veerud, mis tabelis T ning võib olla täiendavaid veerge.

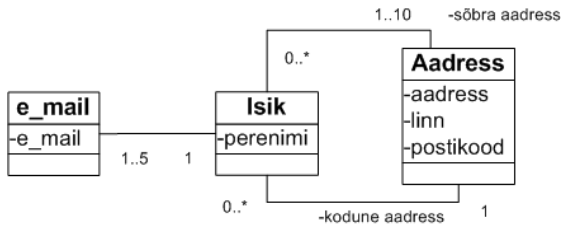
PostgreSQL kui ORDBMS – vähemkasulikud võimalused (2)

- **Mitmene pärimine** (pärimine korraga mitmest tabelist) tüüpimata tabelite vahel
- Tabeli loomine liittüübi põhjal (**tüübitud tabeli** loomine)

Oracle kui ORDBMS

- Kasutaja-defineeritud struktureeritud tüübid.
  - Tüüpidega meetodite seostamine
  - Tabeli veerule kasutaja-defineeritud tüübi määramine
  - Tabeli loomine tüübi põhjal
  - Tüübi põhjal loodud tabeli iga reaga on seotud objekti identifikaator (OID)
  - Tüüpide vahelised pärimisseosed

Näite kontseptuaalne andmemudel



Oracle kui ORDBMS

- Tüübikonstruktorid:
  - VARRAY – ARRAY tüübikonstruktor,
  - Nested table – MULTISET tüübikonstruktor,
  - REF.
- Objektivaated

Struktureeritud tüübi loomine (Oracle)

- CREATE TYPE piirkond_t AS OBJECT (linn VARCHAR2(100), postikood VARCHAR2(10),
- **MAP MEMBER FUNCTION** *piirkond_vordlus* RETURN VARCHAR2) NOT FINAL;
- /

NOT FINAL – saab luua alamtüüpe

piirkond_t

- *piirkond_t* – skalaarne tüüp. Sellel on atribuudid ning meetodid, mille kaudu saab atribuutide poole pöörduda.
- *linn*, *postikood* – atribuudid
- *piirkond_vordlus* – meetod (funktsioon), mis tuleb programmeerijal realiseerida.

Meetodi täpsustamine

- MAP meetod võimaldab vastavat tüüpi väärtuste võrdlemist.
- Meetod tagastab baastüüpi väärtuse, mille põhjal võrdlus läbi viiakse
  - V1>V2 on sama kui V1.map()>V2.map()
  - V1=V2 on sama kui V1.map()=V2.map()
  - ...
  - Võrdlusoperatsioonide läbiviimine *delegeeritakse* baastüübile.
- Objektitüübil võib olla vaid üks MAP meetod.
- Alamtüübil võib olla MAP meetod, kui see on ka ülatüübil.

MAP meetodi näide

```
CREATE OR REPLACE TYPE BODY piirkond_t AS
MAP MEMBER FUNCTION piirkond_vordlus RETURN
   VARCHAR2 IS
   tulem VARCHAR2(1000);
BEGIN
        tulem:= linn || ' ' || postikood;
   RETURN tulem;
END piirkond_vordlus;
END;
/
```


Struktureeritud tüübi loomine alamtüübi loomise kaudu (Oracle)

CREATE TYPE aadress_t **UNDER** piirkond_t (aadress VARCHAR2(100),

CONSTRUCTOR FUNCTION *aadress_t* (linn VARCHAR2, postikood VARCHAR2, aadress VARCHAR2) RETURN SELF AS RESULT,

MEMBER FUNCTION *get_aadress_taishaalikute_arv* RETURN NUMBER) NOT FINAL;

/

Struktureeritud tüübi loomine alamtüübi loomise kaudu (2)

- Alamtüüp võib sisaldada uusi atribuute ja meetodeid.
- Alamtüüp võib muuta ülatüübi meetodi realisatsiooni (üledefineerimine, ingl *overriding*).
- Tüübil võib olla mitu alamtüüpi.
- Tüüpide hierarhias võib olla mitu sama nime kuid erineva signatuuriga (nimi, parameetrite arv/järjekord/tüübid) meetodit (*overloading*).

Konstruktori meetodi täpsustamine

```
CREATE OR REPLACE TYPE BODY aadress_t AS
CONSTRUCTOR FUNCTION aadress_t (linn VARCHAR2, postikood
    VARCHAR2, aadress VARCHAR2) RETURN SELF AS RESULT IS
BEGIN -- Kasutaja loodud konstruktori meetod.
    IF length(postikood)>=5 THEN
            SELF.linn := linn;
            SELF.postikood := postikood;
            SELF.aadress:=aadress;
    ELSE
            RAISE_APPLICATION_ERROR(-20001,'Postikoodis peab olema 5 voi
    rohkem markI');
    END IF;
    RETURN;
END;
```


Meetodi täpsustamine (2)

```
MEMBER FUNCTION get_aadress_taishaalikute_arv
    RETURN NUMBER IS
    arv NUMBER; --Meetod on realiseeritud PL/SQL keeles
BEGIN
    /*Leian aadressis sisalduvate täishäälikute arvu*/
    arv:=nvl(length(aadress)-length(translate(lower(aadress),
    'kaeiouõäöü', 'k')),0);
    RETURN arv;
END;
END;
/
```


Tüübikonstruktorite kasutamine

- CREATE TYPE e_mailid_t AS **VARRAY(5)** OF VARCHAR2(254);

/

- CREATE OR REPLACE TYPE aadressi_tabel_t AS **TABLE OF** aadress_t;

/

Tabeli loomine – veerud on kasutaja loodud tüüpi (Oracle)

- CREATE TABLE Isik_o(isik_id NUMBER(10) PRIMARY KEY, perenimi VARCHAR2(1000), e_mail **e_mailid_t** NOT NULL, kodune_aadress **aadress_t** NOT NULL, soprade_aadressid **aadressi_tabel_t,** *CONSTRAINT isik_o_not_null CHECK (kodune_aadress.aadress IS NOT NULL)*) **NESTED TABLE** soprade_aadressid **STORE AS** soprade_aadressid_nt;

Triger

```
CREATE OR REPLACE TRIGGER trig_i_isik_o BEFORE INSERT OR UPDATE ON Isik_o
FOR EACH ROW
DECLARE
    arv NUMBER;
    sobrad aadressi_tabel_t;
BEGIN
    arv:=Cardinality(:new.soprade_aadressid);
IF (arv IS NULL) THEN --Sõprade aadressid ei tohi olla NULL
    RAISE_APPLICATION_ERROR(-20001,'Peab olema vahemalt yks sober!');
ELSE
    SELECT SET(:new.soprade_aadressid) INTO sobrad FROM Dual; --Korduste eemaldamine
    :new.soprade_aadressid:=sobrad;
    arv:=Cardinality(:new.soprade_aadressid); -- Leian sõprade aadresside arvu
    IF (arv<=1) OR (arv>=10) THEN
     RAISE_APPLICATION_ERROR(-20002, 'Soprade aadresside arv peab olema vahemikus 1 kuni 10!');
    END IF;
END IF;
END;
/
```


Andmete lisamine tabelisse (Oracle)

```
INSERT INTO Isik_o(isik_id, perenimi, e_mail,
    kodune_aadress, soprade_aadressid)
VALUES (1, 'Mets', e_mailid_t('t990999@ttu.ee',
    'metsamees@hot.ee'),
aadress_t('Tallinn', '19086', 'Ehitajate tee 5'),
aadressi_tabel_t(aadress_t('Tallinn', '18532', 'Linnu
    tee 55-1'), aadress_t('Tallinn', '34000', 'Heki tee
    12')));
```

Andmete lisamine tabelisse (Oracle)(2)

- Igal objektitüübil on konstruktori meetod, mille süsteem loob automaatselt.
- Näide pöördumisest konstruktori meetodi poole:
  - aadress_t('Tallinn', '19086', 'Ehitajate tee 5')
- Tüübi defineerimisel on võimalik automaatselt loodav konstruktori meetod üle defineerida.
- Konstruktori meetodi üledefineerimine võimaldab testida andmete vastavust kitsendustele.

Üledefineeritud konstruktori testimine

- UPDATE Isik_o SET kodune_aadress=**aadress_t**('Tallinn', '1908', 'Ehitajate tee 5') WHERE isik_id=1;
- *
- ERROR at line 1:
- ORA-20001: Postikoodis peab olema 5 voi rohkem marki!
- ORA-06512: at "TUD1.AADRESS_T", line 9
- ORA-06512: at line 1

Kitsenduse testimine

- UPDATE Isik_o SET kodune_aadress=**aadress_t**('Tallinn', '19085', NULL) WHERE isik_id=1;
- *
- ERROR at line 1:
- ORA-02290: check constraint (TUD1.ISIK_O_NOT_NULL) violated

Päringu tulemus tabeli põhjal

SELECT * FROM Isik_o;

ISIK_ID	PERENIMI	E_MAIL	KODUNE_AADRESS(LINN, POSTIKOOD, AADRESS)	SOPRADE_AADRESSID(LINN, POSTIKOOD, AADRESS)
1	Mets	E_MAILID_T('t990999@ttu.ee', 'metsamees@hot.ee')	AADRESS_T('Tallinn', '19086', 'Ehitajate tee 5')	AADRESSI_TABEL_T(AADRESS_T('Tallinn', '18532', 'Linnu tee 55-11'), AADRESS_T('Tallinn', '34000', 'Hek i tee 12'))

Probleeme

- Tüüpide deklareerimisel ei saa nende atribuutidele *deklareerida* mingeid kitsendusi.
- Ei saa *deklareerida*, et reaga seotud NESTED tabelis ei tohi olla korduvaid ridu kuid samas võib erinevates NESTED tabelites olla sama rida.
- Ei saa *deklareerida*, et atribuut *soprade_aadressid* on kohustuslik.
- Ei saa *deklareerida* ON DELETE RESTRICT reeglit (Isikut ei saa kustutada, kui on seotud sõprade aadresse).

Probleeme (2)

- Kui mitmel isikul on sama aadressiga sõber, tuleb aadress korduvalt registreerida.
- Kui aadress muutub, tuleb teha muutus mitmes reas.
- Kuidas registreerida aadress, mis pole isikutega seotud?
- Saan muuta tüübikonstruktori/trigeri koodi, kuid sellega ei kaasne automaatne andmete vastavuse kontroll uutele kitsendustele.

Andmete salvestamine

- VARRAY
  - VARRAY andmed salvestatakse sisemisel tasemel RAW või LOBina.
  - Kui VARRAY suurus on alla 4000 baidi, siis salvestatakse koos ülejäänud reaga.
- Nested table
  - Salvestatakse sisemisel tasemel eraldi tabelina.
  - Seose loomiseks 16 baidised süsteemi-genereeritud NESTED_TABLE_ID väärtused. Näide:
    - 00002202086766448B00DFF11D48850204C4F4F5020676644AE0DFF11D48850204C4F4F5020

Andmete kuvamine arusaadavamal kujul (ingl *unnesting*)(Oracle)

- SELECT isik_id, e_mail FROM Isik_o;

ISIK_ID	E_MAIL
1	E_MAILID('t990999@ttu.ee', 'metsamees@hot.ee')

- SELECT i.isik_id, em.* FROM Isik_o i, TABLE(e_mail) em;

ISIK_ID	COLUMN_VALUE
1	t990999@ttu.ee
1	metsamees@hot.ee

Andmete kuvamine arusaadavamal kujul (Oracle) (2)

- SELECT isik_id, soprade_aadressid FROM Isik_o;

ISIK_ID	SOPRADE_AADRESSID(AADRESS, LINN, POSTIKOOD)
1	AADRESSI_TABEL(AADRESS('Linnu tee 55-1', 'Tallinn', '18532'), AADRESS('Heki tee 12', 'Tallinn', '340 00'))

- SELECT i.isik_id, sa.aadress, sa.linn, sa.postikood FROM Isik_o i, TABLE (SET(soprade_aadressid)) sa WHERE linn='Tallinn';

SET – funktsioon, mis eemaldab multihulgast korduvad elemendid

ISIK_ID	AADRESS	LINN	POSTIKOOD
1	Linnu tee 55-1	Tallinn	18532
1	Heki tee 12	Tallinn	34000

Võrdluseks – IBM DB2

- SELECT isik_id, kodune_aadress..linn, kodune_aadress..postikood, kodune_aadress..aadress FROM Isik;

Andmete otsimine tabelist – meetodi kasutamine (Oracle)

- SELECT i.isik_id, i.kodune_aadress, i.kodune_aadress.**get_aadress_taishaalikute_arv()** taishaalikute_arv FROM Isik_o I;

Andmete otsimine tabelist (Oracle) (2)

- Leia selliste isikute identifikaatorid ja sõprade aadresside arv, kelle kohta on teada vähemalt ühe sõbra aadress ja sõprade aadresside kogumis pole korduvaid aadresse:
  - SELECT isik_id, **Cardinality**(soprade_aadressid) AS arv FROM Isik_o WHERE soprade_aadressid **IS NOT EMPTY** AND soprade_aadressid **IS A SET**;

Andmete otsimine tabelist (Oracle) (3)

- Leia selliste isikute identifikaator, kelle sõbra aadress langeb kokku etteantud aadressiga:
  - SELECT isik_id FROM Isik_o WHERE aadress_t('Linnu tee 55-1','Tallinn', '18532') **MEMBER OF** soprade_aadressid;

Andmete otsimine tabelist (Oracle) (4)

- Leidke selliste isikute identifikaatorid, kelle vähemalt üks sõber elab Tallinnas:
  - SELECT isik_id FROM Isik_o WHERE 'Tallinn' IN (SELECT sa.linn FROM TABLE(soprade_aadressid) sa);

Andmete otsimine tabelist (Oracle) (5)

- Leia erinevate aadresside arv:
  - SELECT Count(*) arv FROM (SELECT DISTINCT sa.aadress, sa.linn, sa.postikood FROM Isik_o i, TABLE (soprade_aadressid) sa);

Andmete muutmine struktureeritud tüübiga veerus (Oracle)

- UPDATE isik_o i SET i.kodune_aadress.postikood='19085' WHERE i.kodune_aadress.postikood='19086' AND i.kodune_aadress.linn='Tallinn';

Andmete muutmine struktureeritud tüübiga veerus (IBM DB2)

- UPDATE Isik_o

SET kodune_aadress..postikood='19085'

WHERE kodune_aadress..postikood= '19086' AND kodune_aadress..linn='Tallinn';

- Võimalik on muuta elemendi atribuudi väärtust, pöördudes süsteemi poolt automaatselt genereeritava *mutator* (muutja) meetodi poole.

Kollektsiooni tüüpi andmete muutmine

- Varray ja nested table:
  - uue rea lisamine koos uue kollektsiooniga,
  - kogu kollektsiooni asendamine.
- Nested table:
  - elemendi lisamine kollektsiooni,
  - elemendi muutmine kollektsioonis,
  - elemendi kustutamine kollektsioonist.

Andmete muutmine multihulga tüüpi veerus – näide

- Kustuta isiku 1 sõbra aadress, kus postikood on '34000':
  - DELETE TABLE (SELECT soprade_aadressid FROM isik_o WHERE isik_id = 1) i WHERE i.postikood = '34000';

Kokkuvõte

- ARRAY ja MULTISET tüübikonstruktori abil loodud tüüpide kasutamine Oracle andmebaasi baastabelite veergude tüüpidena.
  - Andmekirjeldus- ja andmekäitluskeele lausete *keerukus suureneb*.
  - Ei saa jõustada enamikke *deklaratiivseid kitsendusi* kollektsiooni kuuluvatele elementidele.
  - *Töökiiruses* võitu ei anna.

Kokkuvõte (2)

- Eksperdid nagu T. Kyte soovitavad *VARRAY* ja *nested table* tüüpe kasutada PL/SQL rutiinides, aga mitte baastabelites.

Parem lahendus

- Andmebaasis kõrge normaliseerituse tasemega baastabelid, kus kollektsiooni tüüpi veerge ei kasutata.
- Objektivaated, kui üks viis andmete esitamiseks.

Objektivaade

- Objektitüübi OT põhjal loodud objektivaate iga rida esitab tüüpi OT kuuluvat väärtust (objekti).
- Objektivaate poolt esitatavad andmed leitakse päringuga tabelite põhjal, mis võivad olla "tavalised" (mitte tüübi põhjal loodud) tabelid.

Näide – tabelid

- CREATE TABLE Isik(isik_id NUMBER(10) PRIMARY KEY, perenimi VARCHAR2(1000) NOT NULL, kodune_aadress NUMBER(10) NOT NULL);
- CREATE TABLE e_mail(e_mail VARCHAR2(254) PRIMARY KEY, isik_id NUMBER(10));

Näide – tabelid (2)

- CREATE TABLE Sobra_aadress(aadress_id NUMBER(10), isik_id NUMBER(10), CONSTRAINT pk_sobra_aadress PRIMARY KEY (aadress_id, isik_id));
- CREATE TABLE Aadress(aadress_id NUMBER(10) PRIMARY KEY, aadress VARCHAR2(100) NOT NULL, linn VARCHAR2(100) NOT NULL, postikood VARCHAR2(10) NOT NULL, CONSTRAINT ak_aadress UNIQUE(aadress, linn, postikood));

Näide – välisvõtmed

- ALTER TABLE Isik ADD (CONSTRAINT fk_Isik_kodune_aadress FOREIGN KEY (kodune_aadress) REFERENCES Aadress(aadress_id));
- ALTER TABLE e_mail ADD (CONSTRAINT fk_e_mail_isik_id FOREIGN KEY (isik_id) REFERENCES Isik(isik_id) ON DELETE CASCADE);

Näide – välisvõtmed (2)

- ALTER TABLE Sobra_aadress ADD (CONSTRAINT fk_Sobra_aadress_aadress_id FOREIGN KEY (aadress_id) REFERENCES Aadress(aadress_id) ON DELETE CASCADE);
- ALTER TABLE Sobra_aadress ADD (CONSTRAINT fk_Sobra_aadress_isik_id FOREIGN KEY (isik_id) REFERENCES Isik(isik_id) ON DELETE CASCADE);

Näide – kitsendused

- Lisaks tuleb trigerite abil realiseerida kitsendused, mille kohaselt ühel isikul võib olla kuni 5 e-maili aadressi ja kuni 10 sõbra aadressi.

Tüübid

- Pean kasutama eelnevalt loodud tüüpe: *aadress_t*, *e_mailid_t*, *aadressi_tabel_t*
- Lisaks loon tüübi:
  - CREATE TYPE Isik_t AS OBJECT (isik_id NUMBER(10), perenimi VARCHAR2(1000), e_mail e_mailid_t, kodune_aadress aadress_t, soprade_aadressid aadressi_tabel_t);

/

Objektivaade

CREATE OR REPLACE VIEW v_Isik OF Isik_t WITH OBJECT IDENTIFIER (isik_id) AS
SELECT isik_id, perenimi,
CAST(MULTISET (SELECT e_mail FROM e_mail E WHERE E.isik_id=I.isik_id) AS e_mailid_t) e_mail,
aadress_t(linn, postikood, aadress) kodune_aadress,
CAST(MULTISET (SELECT linn, postikood, aadress FROM Aadress NATURAL JOIN Sobra_aadress SA WHERE SA.isik_id=I.isik_id) AS aadressi_tabel_t) soprade_aadressid
FROM Isik I INNER JOIN Aadress A ON I.kodune_aadress= A.aadress_id;

Päringu tulemus objektivaate põhjal

```
SELECT * FROM v_Isik;
```

ISIK_ID	PERENIMI	E_MAIL	KODUNE_AADRESS(LINN, POSTIKOOD, AADRESS)	SOPRADE_AADRESSID(LINN, POSTIKOOD, AADRESS)
1	Mets	E_MAILID_T('1990999@ttu.ee', 'metsamees@hot.ee')	AADRESS_T('Tallinn', '19086', 'Ehitajate tee 5')	AADRESSI_TABEL_T(AADRESS_T('Tallinn', '18532', 'Linnu tee 55-1), AADRESS_T('Tallinn', '34000', 'Hek i tee 12'))

Sama tulemus, kui tehes päringut baastabeli *Isik_o* põhjal.

Soovides infot isiku 1 kohta võib programm küsida ühe rea objektivaatest, selle asemel, et küsida mitmeid ridu erinevatest baastabelitest.

Tabeli loomine tüübi põhjal (Oracle)

- CREATE TABLE Isik_o2 **OF Isik_t** (isik_id PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY NESTED TABLE soprade_aadressid STORE AS soprade_aadressid_nt2;

Tüübitud tabel (2)

- Tabel on loodud struktuurse tüübi põhjal
- Igale tüübi atribuudile vastab tabeli veerg
- Iga rida esitab ühte objekti eksemplari
- Iga eksemplariga seotakse unikaalne objekti identifikaator, mis
  - võib olla kasutaja sisestatud,
  - võib olla süsteemi genereeritud.

Relatsiooniline mudel (Date & Darwen)

- Võimalik luua uusi *skalaarseid* tüüpe
  - Võimalik tüüpide loomine pärimise kaudu olemasolevatest tüüpidest
  - Võimalik kasutada tüüpide loomisel deklaratiivseid kitsendusi (pole võimalik SQLis)
- Võimalik kasutada *tüübigeneraatoreid* (TUPLE, RELATION)
- Kasutaja loodud tüüpe saab kasutada kõikjal, kus andmebaasis tüüpe kasutatakse

Relatsiooniline mudel (Date & Darwen) (2)

- Mis on *ebaotstarbekas* ning lisab vaid *keerukust*? (seega relatsiooniline mudel ei toeta)
  - Relatsiooniliste muutujate loomine struktuursete (skalaarsete) tüüpide põhjal
  - Relatsioonilise muutuja loomine pärimise kaudu olemasolevast relatsioonilisest muutujast
  - Viitade (pointer) kasutamine relatsioonide vaheliste seoste loomiseks

Näide

- Järgnevalt demonstreeritakse, kuidas lahendada eelnevalt käsitletud objektivaate loomise näidet *Kolmanda Manifesti* põhimõtteid järgivas relatsioonilises andmebaasisüsteemis
- Andmebaasisüsteem – Rel 2.07
- Andmebaasikeel – Tutorial D

Relatsiooniliste baasmuutujate loomine

- VAR Isik BASE RELATION {isik_id INTEGER, perenimi CHAR, aadress_id INTEGER} KEY {isik_id};
- VAR e_mail BASE RELATION {e_mail CHAR, isik_id INTEGER} KEY {e_mail};
- VAR Sobra_aadress BASE RELATION {aadress_id INTEGER, isik_id INTEGER} KEY {aadress_id, isik_id};
- VAR Aadress BASE RELATION {aadress_id INTEGER, aadress CHAR, linn CHAR, postikood CHAR} KEY {aadress_id} KEY {aadress, linn, postikood};

Välisvõtme kitsenduste deklareerimine

- CONSTRAINT e_mail_Isik_foreign_key e_mail {isik_id} = (e_mail JOIN Isik) {isik_id};
- CONSTRAINT sobra_aadress_Isik_foreign_key Sobra_aadress {isik_id} = (Sobra_aadress {isik_id} JOIN Isik) {isik_id};
- CONSTRAINT sobra_aadress_Aadress_foreign_key Sobra_aadress {aadress_id} = (Sobra_aadress {aadress_id} JOIN Aadress) {aadress_id};

Täiendavate kitsenduste deklareerimine

- Isikul võib olla kuni 5 e-maili
  - CONSTRAINT Isik_has_at_most_5_e_mail (Count((SUMMARIZE e_mail PER (Isik {isik_id}) ADD (Count() AS cnt)) WHERE cnt>5)=0);
- Isikul võib olla kuni 10 seotud sõbra aadressi
  - CONSTRAINT Isik_has_at_most_10_Sobra_aadress (Count((SUMMARIZE Sobra_aadress PER (Isik {isik_id}) ADD (Count() AS cnt)) WHERE cnt>10)=0);

Relatsiooniliste muutujate väärtustamine testandmetega

- INSERT Aadress RELATION {
  TUPLE {aadress_id 1, aadress "Ehitajate tee 5", linn "Tallinn", postikood "19086"},
  TUPLE {aadress_id 2, aadress "Linnu tee 55-1", linn "Tallinn", postikood "18532"},
  TUPLE {aadress_id 3, aadress "Heki tee 12", linn "Tallinn", postikood "34000"}};
- INSERT Isik RELATION {
  TUPLE {isik_id 1, perenimi "Mets", aadress_id 1}};
- INSERT e_mail RELATION {
  TUPLE {isik_id 1, e_mail "t990999@ttu.ee"},
  TUPLE {isik_id 1, e_mail "metsamees@hot.ee"}};
- INSERT Sobra_aadress RELATION {
  TUPLE {isik_id 1, aadress_id 2},
  TUPLE {isik_id 1, aadress_id 3}};

multiple assignment – ühe lausega väärtus mitmele muutujale

Vaate loomine

- VAR v_Isik VIRTUAL ((((Isik JOIN e_mail) GROUP ({e_mail} AS e_mailid)) JOIN Aadress WRAP ({aadress, linn, postikood} AS kodune_aadress)) {ALL BUT aadress_id} JOIN Sobra_aadress JOIN Aadress) {ALL BUT aadress_id} GROUP ({aadress, linn, postikood} AS soprade_aadressid);

Päring vaate põhjal

v_Isik

isik_id INTEGER	perenimi CHARACTER	e_mailid RELATION	kodune_aadress TUPLE			soprade_aadressid RELATION		
		e_mail CHARACTER	aadress CHARACTER	linn CHARACTER	postikood CHARACTER	aadress CHARACTER	linn CHARACTER	postikood CHARACTER
1	Mets	t990999@ttu.ee	Ehitajate tee 5	Tallinn	19086	Linnu tee 55-1	Tallinn	18532
		metsamees@hot.ee				Heki tee 12	Tallinn	34000

- Atribuut *kodune_aadress* on korteeži tüüpi
- Atribuudid *e_mailid* ja *soprade_aadressid* on relatsiooni tüüpi

Mõningad teema põhimõisted