Proprietăți obiectual-relaționale în standardul SQL

prof. dr. ing. Mircea Petrescu

Tipuri construite interne (build-in)

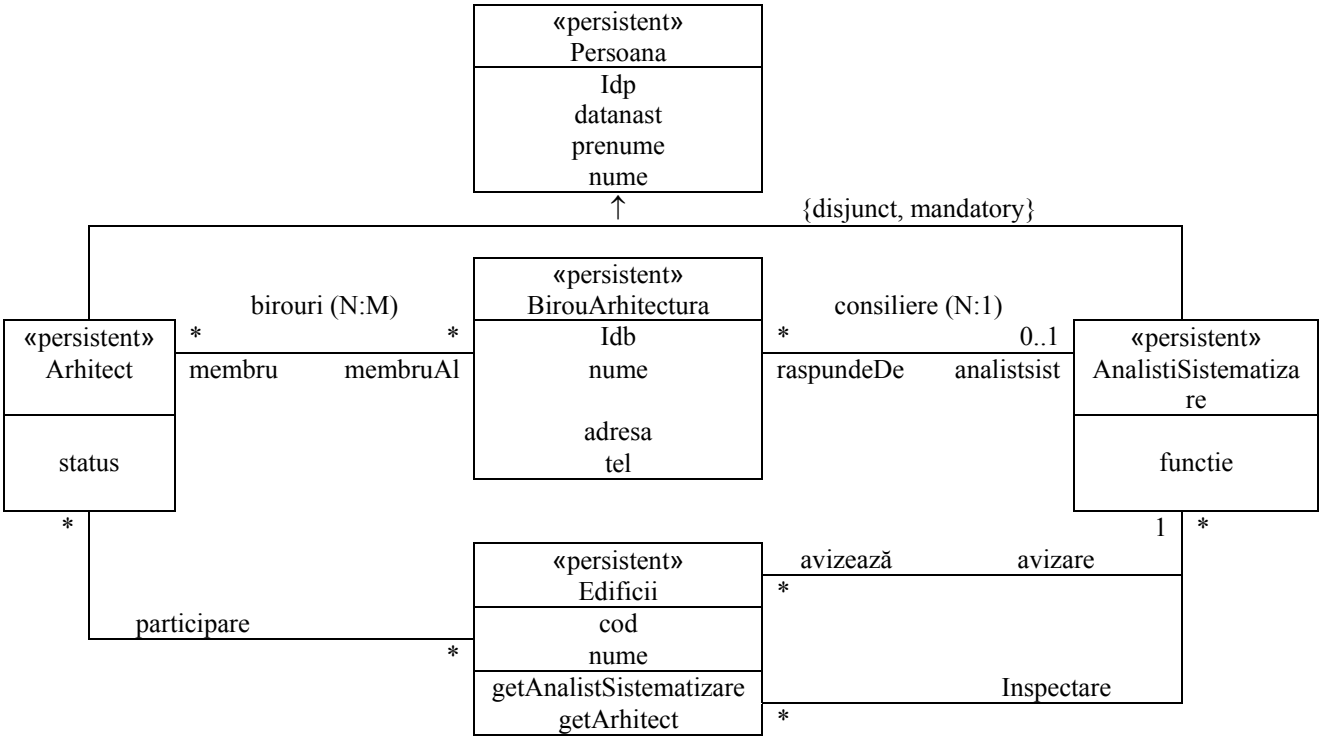
Din faza inițială a existenței sale, SQL a permis utilizarea tipurilor atomice pentru definirea coloanelor, variabilelor şi parametrilor. Tot acolo a fost introdusă o clasă de tipuri de date numite tipuri construite. Acestea sunt tipuri definite de utilizator (UDT), tipuri referință, tipuri linie (rând), precum şi tipuri colecție.

Tipurile construite pot conține mai mult decât o valoare. Din cele patru componente ale familiei tipurilor construite, ultimele două (tipurile rând şi tipurile colecție) sunt interne (build-in), iar primele două (UDT şi referință) pot fi folosite de utilizator pentru a defini, prin extindere, tipuri noi, pe lângă cele interne.

Tipuri linie (rând)

În accepțiunea obişnuită folosită în contextul modelului relațional, o linie (rând) este o colecție nevidă de valori, în care tipul fiecărei valori corespunde unei definiții de coloană dintr-o tabelă. După cum ştim, o tabelă relațională convențională (relație) este formată din linii (rânduri) cu proprietatea că fiecare valoare de coloană, în fiecare linie, trebuie să fie atomică, aceasta fiind definiția FN1.

Standardul SQL lărgeşte cerințele FN1, prin introducerea tipului linie (rând), care admite ca o linie să conțină o altă linie, aceasta fiind introdusă ca o valoare de coloană. Se observă că tipul linie (rând) este asemănător cu structurile de înregistrare din limbajele de programare şi cu tipul struct din limbajul obiectual ODL.




```
create table birouArhitectura(
      idb varchar(10),
```

```
nume varchar(50) not null,
adresa row (strada varchar(30), cladirea varchar(5), camera varchar(5)),
analistsistematizare varchar(11) references arhitecti(idp),
primary key (idb));
```

Constructorul de linie (rând) se foloseşte pentru a atribui valori câmpurilor unei linii. Valorile folosite de constructorul de linie pot fi o listă de valori, sau rezultatul unei interogări. În fiecare din aceste cazuri, tipurile valorilor trebuie să corespundă tipurilor de câmp folosite în definiția tipului linie. Exemplu:

```
insert into birouArhitectura values(
'BA101',
'Birou Arhitect Univ',
row('Spaiul Independentei', 'Cladire centrala', 'cam 301'),
'DA110');
```

În comanda de mai sus, adresa biroului de arhitectură este reprezentată ca o coloană în tabelă, însă această adresă este o valoare nonatomică formată din trei valori de şiruri de caractere separate. De asemenea, DA are sensul de Director Arhitect.

Valorile unui tip (rând) pot fi extrase folosind notația „cu punct" pentru a realiza accesul la câmpurile individuale care sunt părți ale coloanei. De exemplu:

```
select b.adresa.strada, b.adresa.cladirea, b.adresa.camera
from birouArhitectura b
where b.nume = 'Birou Arhitect Univ';
```

Tablouri de colecții

Valorile nonatomice din schemele relaționale pot fi reprezentate şi prin tipul colecție. În general, o colecție poate fi o structură de date din categoria mulțimilor, listelor, multiset-urilor sau tablourilor. Standardul SQL permite, în forma sa actuală, numai tipul tablou.

O coloană a unei tabele poate fi definită (specificată) ca un tablou, adăugând la tipul de date al coloanei cuvântul cheie array. La rândul său, acest cuvânt cheie este urmat de numărul maxim de elemente pe care le poate conține, înscris între paranteze drepte. Accesul la prima poziție dintr-o tabelă este realizat cu o valoare de index de unu.

În exemplul care urmează se arată cum pot fi memorate direct identificatoarele membrilor biroului de arhitectură în interiorul fiecărei linii (rând) a tabelei (relației) birouArhitectura, folosind atributul cu numele membri:

```
create table birouArhitectura(
Idb varchar(10),
nume varchar(50),
adresa row(strada varchar(30), cladirea varchar(5), camera varchar(5)),
analistsistematizare varchar(11) references architect(Idp),
membri varchar(11) array[50] references architect(Idp),
primary key (Idb));
```

Constructorul de tip array este folosit pentru a rezerva spatiu structurii tablou şi, de asemenea, serveşte la atribuirea de valori elementelor tabloului. Pentru a inițializa structura tabloului membrii la o valoare vidă se foloseşte instrucțiunea insert cu un constructor de tip array fără elemente:

```
insert into birouArhitectura values(
'BA101',
'Birou Arhitect Univ',
```

```
row('Spl Independentei', 'Cladire centrala', 'Cam 301'), 'AS110',
array[]);
```

În continuare, structurii tablou i se pot adăuga identificatoarele specifice ale membrilor biroului de arhitectură, cu ajutorul constructorului de tablou, în cadrul unei instrucțiuni update. Pentru atribuirea de valori primelor trei poziții ale structurii tablou membrii se foloseşte aserțiunea:

```
update birouArhitectura
set membri=array['…','…','…']
where nume='Birou Arhitect Univ';
```

Atribuiri de valori ale pozițiilor individuale în interiorul unei structuri tablou se pot efectua, de asemenea, cu ajutorul unei valori de index specifice:

```
update birouArhitectura
set membri[4]='…'
where nume='Birou Arhitect Univ';
```

Pentru a realiza accesul la o poziție anumită într-un tablou, se foloseşte un index. De pildă, pentru a extrage identificatorul celui de al doilea element al unui birou de arhitectură folosim instrucțiunile:

```
select membri[2]
from birouArhitectura
where nume='Birou Arhitect Univ';
```

Pentru determinarea dimensiunii (lungimii) unui tablou se foloseşte funcția cardinality. Această funcție este utilă pentru realizarea iterațiilor în interiorul unui tablou, în rutinele SQL. De exemplu, pentru a găsi lungimea curentă a tabloului membri, admițând că în tabela birouArhitectura se foloseşte un cursor C, este introdusă comanda cardinality(C.membri).

Folosirea generală a tablourilor constă în reprezentarea atributelor multi-valoare. De asemenea, tablourile sunt utile în modelarea părții multi a asocierilor 1:N sau M:N, inclusiv în cazul limbajului de definire ODL.

Tipuri definite de utilizator (UDT)

Termenul UDT este standard în SQL, având acelaşi înțeles ca termenul „tip abstract de date". În legătură cu acest ultim concept, ne reamintim că utilizatorul defineşte tipuri noi care au o anumită formă a structurii interne, fiind de asemenea definite metode care descriu comportarea acelor tipuri. Reprezentarea internă a unui tip este încapsulată de comportarea acestuia, ceea ce înseamnă că implementarea internă a tipului de date este ascunsă față de lumea exterioară. Această implementare se poate schimba, fără a influența modul în care un utilizator interacționează cu tipul de date.

În standardul SQL, tipurile definite de utilizator sunt însă întrucâtva diferite față de definiția strictă a tipurilor abstracte de date. Diferența provine în special din faptul că atât toate atributele interne ale unui UDT, cât şi metodele asociate, sunt publice. Aceste elemente nu pot fi „mascate" ca fiind protejate sau private, aşa cum se procedează în limbaje ca Java sau C++.

Folosirea atributelor publice şi a metodelor dă posibilitatea ca structura internă a unei instanțe a tipului introdus de utilizator să fie integrată printr-un limbaj ca SQL. Virtutea principală a UDT stă în faptul că ele permit proiectanților de baze de date să definească tipuri noi de date, orientate către aplicație, în afară de tipurile atomice şi construite din categoria „build-in", iar apoi să folosească noile tipuri în definirea de tabele (relații). Desigur, cu ajutorul UDT pot fi create tabele în mediul relațional.

Sintaxa SQL pentru crearea tipurilor definite de utilizator (UDT)

```
create table <user_defined_type_body>
<user_defined_type_body> ::= <user_defined_type_name>
[under <user_defined_type_name>]
[as <representation>]
[[not] <instantiable>]
[[not] final]
[ref is system generated | ref using <predefined_type> | ref from
<attribute_name> [{, attribute_name} …]]
[<method_specification_list>]
<representation> ::= <predefined_type> | <member_list>
<member_list> ::= (<attribute_definition>) [{, <attribute_definition>} …]
<attribute_definition> ::= <attribute_name> {<data_type> | <collection_type>}
[<reference_scope_ check>] [default <default_value>]
<data_type> ::= <predefined_type> | <reference_type>
<collection_type> ::= <data_type> array [unsigned_integer] /* [] part of syntax
*/
<method_specification_list>        ::=        <method_specification>        [{,
<method_specification>} …]
<method_specification>        ::=        <partial_method_specification>        |
<overriding_method_specification>
<overrinding_method_specification> ::= overriding <partial_method_specification>
<partial_method_specification>    ::=    [constructor]    method    <method_name>
<SQL_parameter_declarations> returns <data_type>
```

Tipuri distincte

Folosesc pentru asocierea unei înțelegeri speciale cu un tip atomic existent. De fapt, definesc o formă nouă a unui tip atomic. Exemplu:

```
create type varsta as integer final;
create type greutate as integer final;
create table persoana(
Idpersoana varchar(3),
Varstapersoana varsta,
Greutatepersoana greutate,
primary key (Idpersoana));
```

Cuvântul cheie final reprezintă elementul de sintaxă pentru tipurile distincte în standardul SQL actual, indicând faptul că nu poate fi definit un subtip al tipului distinct. Mai observăm că odată definite valorile varsta şi greutate de mai sus nu pot fi comparate. Aceste valori nu pot fi „amestecate” cu tipul integer normal, întrucât varsta, greutate si integer sunt, conceptual, tipuri diferite. Totuşi, remarcăm că folosirea în aceeaşi construcție a tipurilor distincte şi a tipurilor atomice pe care ele se bazează este posibilă – dar numai atunci când folosim (adăugăm) funcția cast. Exemplu:

```
select Idpersoana
from persoana
where cast(varstapersoana as integer)*2 < cast(greutate persoana as integer);
```

Tipuri structurate

Sunt compuse din câteva componente interne, fiecare din acestea putând fi de un tip diferit. O instanță a unui tip structurat este o valoare compusă, deoarece tipul conține mai multe componente. Exemplu:

```
create type adresaUdt as(
strada varchar(90),
cladirea varchar(5),
camera varchar(5)) not final;
```

Mai sus, clauza not final indică posibilitatea definirii de sub-tupluri ale tipului structurat. Menționăm că atributele tipului structurat pot aparține unor tipuri build-in atomice, construite sau chiar UDT.

Tipul adresaUdt definit poate, la rândul său, contribui la definirea tipului coloanei adresa în tabela birouArhitectura:

```
create table birouArhitectura(
Idb varchar(10),
nume varchar(50) not null,
adresa adresaUdt,
analistsist varchar(11) references analistsistematizare(Idp),
membri varchar(11) array[50] references architect(Idp),
primary key (Idp));
```

Metode interne pentru tipurile structurate

Într-un tabel anterior, am arătat, mai înainte, în contextul rutinelor menționate (invocate) în cadrul SQL, că sunt trei tipuri de proceduri stocate: funcții, proceduri şi metode. Metodele sunt asociate în special cu tipurile structurate, ele fiind de fapt funcții strâns legate cu definirea acestor tipuri. Ca urmare, tipurile structurate permit încapsularea – un tip de acest fel fiind manipulat (prelucrat) numai în metode definite de el. În standardul SQL, metodele pot fi definite ca proceduri. O excepție – Oracle, care admite definirea metodelor ca funcții şi proceduri.

Tipuri de metode interne (build-in) pentru tipurile structurale de date: funcția constructor, funcția observer (observator), funcția mutator. Aceste metode sunt conținute automat în definirea tipurilor. Funcția constructor are acelaşi nume ca şi tipul, şi este folosită în crearea de instanțe ale tipului, fiind invocată prin expresia new. Funcția observer este folosită pentru extragerea valorilor unui tip structurat; pentru fiecare atribut al tipului structurat există o funcție observer, cu acelaşi nume ca atributul. Funcția mutator este folosit pentru modificarea valorilor atributelor unui tip structurat. Funcțiile observer si mutator sunt menționate (invocate, de fapt) prin notația „punct” („dot”): (Variabilă.NumeFuncție). Această notație stă în locul notației convenționale (NumeFuncție (parametrii)), folosită numai pentru funcțiile care nu sunt metode.

Exemplu – rutină SQL prin care se crează o instanță de tipul adresaUdt, folosind funcțiile constructor şi mutator:

```
begin
      declare adr adresaUdt
      /* invocare functie constructor */
      set adr = new adresaUdt();
      /* invocare functie mutator */
      set adr.strada = 'Spl Independentei';
      set adr.cladirea = 'Cladire Centrala';
      set adr.camera = 'cam 301';
      insert into birouArhitectura values(
            'BA101',
            'Birou Arhitect Univ',
            adr, /* initializare adresa */
            'AS110',
            Array[]);
end.
```

Pentru a extrage valori de atribute ale tipului structurat se foloseşte funcția observer, ca în exemplul:

```
select nume, c.adresa.strada, c.adresa.cladirea, c.adresa.camera
from birouArhitectura c
where nume = 'Birou Arhitect Univ';
```

Metode definite de utilizator pentru tipurile structurale

Utilizatorul defineşte semnătura fiecărei metode, prin introducerea numelui metodei, a numelor parametrilor si a tipurilor acestora. Exemplu: definirea metodei cu numele suma pentru tipul structurat cu numele treiNumere. Semnătura arată că această metodă nu are parametrii expliciți definiți. Metoda are ca rezultat o valoare de tip integer. De obicei însă, fiecare metodă are un parametru implicit, care este instanța tipului de date pentru care se defineşte metoda. Valoarea parametrului implicit este obținută în implementarea metodei prin cuvântul cheie self. Iată definiția metodei:

```
create type treiNumere as
(unu integer,
doi integer,
trei integer) not final;
method suma() returns integer;
create method suma() returns integer for treiNumere
begin
      return self.unu + self.doi + self.trei;
end.
```

Utilizatorul / programatorul poate ignora funcția constructor a unui tip structurat. Deoarece funcția constructor definită de sistem nu admite parametrii, prin ignorarea funcției constructor se realizează o formă diferită a funcției, care poate fi folosită pentru a preciza valorile atributelor specifice în momentul în care este creată o instanță a tipului de date.

În continuare, este prezentat un exemplu privind ignorarea funcției constructor a tipului adresaUdt. Potrivit sintaxei SQL pentru crearea UDT, cuvântul cheie overriding trebuie specificat neapărat, pentru a indica faptul că metoda suprascrie o funcție existentă:

```
create type adresaUdt as(
strada varchar(30),
cladirea varchar(5),
camera varchar(5)) not final;
overriding constructor method /**/
      adresaUdt(strada varchar(30), cladirea varchar(5), camera varchar(5))
      returns adresaUdt;
create method adresaUdt(str varchar(30), clad varchar(5), cam varchar(5))
      returns adresaUdt for adresaUdt
begin
      set self.strada = str;
      set self.cladirea = clad;
      set self.camera = cam;
end.
```

Cuvântul cheie constructor trebuie specificat mai sus deoarece metoda care se ignoră este, în acelaşi timp, o funcție constructor. Desigur, numele metodei este acelaşi ca numele funcției constructor definită de sistem (adresaUdt). Definiția metodei conține specificarea parametrilor şi tipurilor acestora. Implementarea metodei arată modul în care se folosesc parametri pentru a atribui valori unei instanțe prin funcții mutator. De observat că metoda extrage self ca o valoare, care este instanța modificată de tipul adresaUdt.

Funcția constructor definită de utilizator poate fi acu folosită pentru a crea o instanță nouă de tipul adresaUdt şi, în acelaşi timp, de a introduce valori ale atributelor acesteia:

```
declare adr adresaUdt;
set adr = new adresaUdt('Spl Independentei', 'Cladire Centrala', 'Cam 301');
```

Tabele tipizate

În standardul SQL, o instanță a unui UDT este o valoare. Pentru a crea conceptul de obiect, ca în tehnologia bazelor de date orientate obiect, trebuie folosit un UDT împreună cu o tabelă tipizată. Tabela tipizată este o formă nouă de tabelă în standardul SQL, asociată întotdeauna cu un anumit tip structurat. Pentru fiecare atribut al tipului structurat pe care se sprijină, o tabelă tipizată are o coloană. În plus, va avea o coloană cu „auto-referire", care conține un identificator unic de obiect pentru fiecare linie (rând) al tabelei. Acest identificator se numeşte referire.

Când pentru definirea unei tabele tipizate se foloseşte un tip structurat, o instanță a acestui tip este văzută ca un obiect, al cărui identificator este dat de coloana cu „auto referire". Spre deosebire de identificatoarele de obiecte folosite în bazele de date orientate obiect, un identificator de obiect este unuic numai în contextul unei tabele tipizate specifice. Deci două tabele tipizate pot avea linii cu valori cu „auto-referire" identice.

Pentru a exemplifica folosirea tabelelor tipizate, să admitem că dorim implementarea clasei Edificii din diagrama de clase Persoana-Arhitect-Edificii, ca o tabelă tipizată. Mai întâi, definim tipul structurat edificiiUdt:

```
create type edificiiUdt as(
cod varchar(3),
nume varchar(40))
instantiable not final ref is system generated;
```

În codul de mai sus, edificiiUdt defineşte atributele cod şi nume. Observăm însă prezența a două clauze, instantiable şi ref, care au fost introduse anterior, prin sintaxa SQL pentru crearea UDT.

Clauza instantiable arată că pentru tipul creat există o funcție constructor şi astfel utilizatorul (prgramatorul) poate crea direct instanțe ale acestui tip. Dacă însă tipul creat este specificat ca fiind „ne-instanțiabil", prin clauza not-instantiable, nu va exista o funcție constructor pentru acest tip. Folosirea ultimei clauze, de ne-instanțiere, are sens numai pentru un tip care posedă un sub-tip, caz în care instanțele tipului sunt create numai la nivel de sub-tip. Ierarhiile de tipuri se vor examina mai târziu. Un tip structurat folosit împreună cu o tabelă tipizată trebuie întotdeauna specificat ca „instanțiabil".

Clauza ref permite programatorului să specifice modul (mijloacele) de generare a valorii pentru referirea unui obiect. Sunt trei forme a clauzei ref:

a) ref – operațiunea de referire este generată de sistem; în acest caz, sistemul de bază de date este răspunzător de generarea unei referiri unice de obiect pentru instanțele tipului.
b) ref using – arată că referirea este generată de programator. Aceasta trebuie să introducă o referire unică pentru fiecare instanță a tipului, valoarea referirii fiind de tipul indicat în clauză.
c) ref from – arată că programatorul este cel care trebuie să dea lista atributelor tipului structurat, care va fi folosită pentru a obține o referire unică de obiect.

În contextul de aici, se utilizează numai referirile generate de sistem, ca cea mai adecvată pentru înțelegerea folosirii obiectelor.

Sintaxa SQL pentru crearea tabelelor tipizate

După ce a fost definit UDT, ca mai sus, se crează o tabelă tipizată corespunzătoare structurii acesteia. Iată sintaxa pentru crearea tabelelor tipizate:

```
create table <table_name> of <user_defined_type_name> [under <supertable_name>]
[<table_element_list>]
<table_element_list> ::= (<table_element> [{, <table_element>} …])
<table_element> ::= <table_constraint> | <self_referencing_column_specification>
| <column_options>
<self_referencing_column_specification>          ::=          ref          is
<self_referencing_column_name> <reference_generation>
<reference_generation> ::= system generated | user generated | derived
<column_options> ::= <column_name> with options <column_option_list>
<column_option_list> ::= [scope <table_name> [<reference_scope_check>]] [default
<default_value>] [<column_constraint> …]
```

Să definim, pe această bază, tabela tipizată edificii:

```
create table edificii of edificiiUdt(
primary key (cod),
ref is edificiiID system generated);
```

Clauza of, mai sus, arată pe ce tip structurat se bazează tabela tipizată. Ca urmare, tabela va avea, în mod automat, coloane care corespund atributelor tipului structurat. Remarcăm că definiția tabelelor tipizate acceptă aceleaşi restricții pentru tabelă şi coloane, ca cele din cazul tabelelor relaționale convenționale. De pildă, în definiția tabelei edificii, atributul cod al tuplului edificiiUdt este luat ca cheie primară. De asemenea, se pot specifica restricții ca unique sau not null, în formatul restricțiilor de tabelă sau de coloană. În definiția unei tabele tipizate sunt indicate numai atributele cu restricții.

Potrivit sintaxei de mai sus, definiția unei tabele tipizate trebuie să repete specificația generării referirilor într-o manieră consistentă cu specificația generării referirilor tipului structurat: system generated, user generated, derived.

De asemenea, prin clauza ref is trebuie atribuit un nume coloanei cu „auto-referire”. Acest nume (în cazul nostru edificiiID) poate fi folosit pentru „manipularea” coloanei cu „auto-referire” în cazul referirilor definite de utilizator. Pentru a avea acces la valoarea de „auto-referire” se foloseşte coloana cu „auto-referire”.

Pentru a introduce linii (rânduri) în o tabelă tipizată – se foloseşte comanda insert, în mod concurențial. De exemplu:

```
insert into edificii values(
'SIC', 'Stiinta si Ingineria Calculatoarelor');
insert into edificii values(
'IM', 'Inginerie Mecanica');
```

Liniiile (rândurile) care se repetă în tabela tipizată edificii vor fi:

(coloana cu „auto-referire”) edificiiID	cod	nume
10287534556	SIC	Stiinta si Ingineria Calculatoarelor
90324854948	IM	Inginerie Mecanica

Valorile pentru coloana cu „auto-referire” sunt generate de sistem. Dacă referirea este definită de utilizator, comanda insert trebuie să conțină valoarea pentru coloana cu „auto-referire”. Dacă referirea este derivată din atributele tipului, restricția primary key sau restricțiile unique şi not null pot fi prezente în definiția tabelei, pentru a asigura o valoare unică de referire pentru atribute.

Tipurile referire

Anterior, a fost examinată folosirea coloanei cu „auto-referire”, care este, de fapt, un identificator intern de obiect, sau referire, către o linie (rând). Această referire este un tip de date numit „tip referire” („reference type”), sau ref. Sintaxa este:

```
<reference_type>   ::=   ref   (<referenced_type>)   [<scope_clause>]   [array
[usingend_integer]] [reference_scope_check] /* [] part of syntax */
<scope_clause> ::= scope <table_name>
<refenced_type> ::= <user_defined_type_name>
<reference_scope_check> ::= references are [not] checked [on delete <action>]
```

Sintaxa pentru definirea tipurilor referire

Ca exemplu, să considerăm din nou diagrama de clase Persoana-Arhitect-Edificii. Între BirouArhitectura şi AnalistSistematizare avem o asociere (legătură) 1:N. De asemenea, între BirouArhitectură şi Arhitect avem o asociere M:N. Putem acum redefinii tipul BirouArhitectUdt, pentru a folosi tipuri „referire” ca un mijloc de descriere a asocierilor cu obiectele AnalistSistematizare şi arhitect:

```
create type birouArhitectUdt as(
Idb varchar(10),
nume varchar(50) not null,
adresa adresaUdt,
tel varchar(12),
analistSistemtizare ref(analistSistematizareUdt),
membri ref(arhitectUdt) array[50])
instantiable not final ref is system generated;

create table birouArhitecturaUdt(
primary key (Idb),
ref is birouArhitecturaID system generated);
```

Exemplul de mai sus – numai pentru a ilustra crearea tipurilor „referire”. Verificarea referirilor şi interogarea acestor tipuri – mai târziu.