

Capitolul 6 - Baze de date obiect-relaționale

- Caracteristicile modelului obiect-relațional
- Tipuri de date definite de utilizator și instanțele acestora (obiecte)
- Metodele tipurilor definite de utilizator; constructori
- Categorii de obiecte: de coloană (*column object*) și de linie (*row object*)
- Operații cu obiecte și colecții de obiecte
- Identificatorii obiectelor (OID)
- Referințe la obiecte (REF)
- Moștenirea tipurilor de date
- Colecții de date în modelul OR:
 - vectori de elemente (varray)
 - tabele imbricate (nested tables)
- Proiectarea bazelor de date obiect-relaționale
 - Exemplu – baza de date PurchaseOrder (PO) în sistemul Oracle
- Caracteristici obiect-relaționale în alte SGBD-uri

Motivația pentru modelul obiect-relațional (OR)

- Reluăm deficiențele modelului relațional:
 - nu oferă tipuri de date predefinite (built-in) complexe
 - nu suportă tipuri definite de utilizator
 - admite numai forme normale ale relațiilor – cel puțin în prima forma normală, care impune memorarea numai de valori atomice și scalare ale atributelor; pentru valori multiple ale unui atribut, trebuie creată o nouă relație
- Deficiențele modelului obiect-orientat de baze de date:
 - complexitate mare a tipurilor de date
 - întrepătrundere între proiectul bazei de date și aplicațiile bazei de date
 - lipsa unui limbaj standardizat atât de concis și productiv cum este SQL
- Modelul obiect-relațional
 - combină avantajele modelului relațional cu cele ale modelului obiect
 - încercă să elimine cât mai multe din deficiențele fiecăruia dintre modele
- Tranziția de la modelul relațional la modelul obiect relațional → în mod treptat, pentru a micșora riscul de piață: Oracle, IBM, Microsoft

Caracteristicile modelului obiect-relațional (OR)

■ Caracteristicile modelului OR:

1. Tipuri de date definite de utilizator (*UDT* - user defined types, sau tipuri abstracte de date - *ADT* abstract data types) → corespund claselor (OO)
 - se folosește modelul de date relațional, dar attributele pot avea ca valori obiecte, instanțe ale unor tipuri definite de utilizator
 - tipurile de date definite de utilizator sunt clase → încapsulează attribute și metode; metodele se pot defini în diferite limbaje (PL/SQL, Java, C, C++)
 - permite moștenirea tipurilor (claselor) și polimorfismul
 - obiecte → instanțe ale tipurilor (claselor); se pot defini identificatori ai obiectelor (OID – Object Identifier) și referințe la obiecte (REF)
 2. Tipuri de date pentru obiecte mari – BLOB (*bits large objects*), CLOB
 3. Colecții de date: vectori de elemente și tabele imbricate
 - permite utilizarea unei colecții ca valoare a unui atribut → elimină restricția de formă normală FN1
- ## ■ Suport pentru standardul SQL:
- SGBD-urile relaționale suportă standardul SQL-92 (SQL-2) (sau precedente)
 - SGBD-urile obiect-relaționale (OR) suportă standardul SQL-99 (SQL-3) sau versiuni ulterioare (SQL:2003, SQL:2005, SQL:2008)

Standarde SQL

- Marea majoritate a SGBD-urilor relaționale și obiect-relaționale suportă una din versiunile standardului SQL, care oferă instrucțiuni pentru toate operațiile cu bazele de date
- SQL este standardizat de două comitete internaționale recunoscute de majoritatea producătorilor și utilizatorilor de baze de date: ANSI (*American National Standardization Institute*) și ISO (*International Standardization Office*), afiliat la IEC (*Int. Electrotechnical Commission*)
- Fiecare standard SQL este publicat de ambele comitete, cu denumiri conform propriilor convenții, dar identice din punct de vedere tehnic
- Primele standarde SQL (până la SQL-92 inclusiv) prevedeau instrucțiuni pentru modelul relațional, fără suport pentru tipuri de date complexe (definite de utilizator) sau pentru controlul ordinii de execuție a instrucț.
- Standardele SQL ulterioare (SQL-99, SQL:2003, SQL:2005, SQL:2008) definesc modelul obiect-relațional de baze de date, cu suport pentru tipuri de date complexe și controlul ordinii de execuție a instrucțiunilor

Standardul SQL:2008 (1)

- Standardul SQL:2008 conține mai multe părți, cu denumirile:
 - ANSI/ISO/IEC 9075:2008, "Database Language SQL": Parts:
1 ("SQL/Framework"), 2 ("SQL/Foundation"), 3 ("SQL/CLI"), 4 ("SQL/PSM"),...
9 ("SQL/MED"), 10 ("SQL/OLB"), 11("SQL/Schemata"),...
 - ANSI/ISO/IEC 9075-14:2008, "Database Language SQL",Part 14("SQL/XML")
- Grade de conformitate (*compliance*) al SQL implementat cu standardul:
 - Suport complet (*full support*) – conformitate completă ca sintaxă și semantică
 - Suport parțial (*partial support*) – conf. numai pentru o parte din standard
 - Suport sporit (*enhanced support*) – conf. și funcționalități suplimentare
 - Suport echivalent (*equivalent support*) – conf. semantică, diferențe sintactice
 - Suport similar (*similar support*) – diferențe semantice și sintactice, dar funcționalitate similară
- SGBD-urile implementează diferite caracteristici SQL cu diferite grade de conformitate cu standardele existente:
 - Oracle SQL oferă *full support* (pentru toate caracteristicile) SQL:2008
 - Alte SGBD-uri oferă suport numai pentru anumite caracteristici SQL:2008

Standardul SQL:2008 (2)

- Partea obligatorie de conformitate cu standardul pentru SGBD-urile relaționale este numită Core SQL:2008 și este cuprinsă în SQL:2008 Part 2 (*Foundation*) și Part 11 (*Schemata*)
- Instrucțiunile de control al ordinii de execuție sunt cuprinse în SQL:2008 Part 4 (PSM – *Persistent Stored Modules*); acestea permit scrierea și execuția modulelor stocate (proceduri, funcții, triggeri)
- Astfel de instrucțiuni sunt prevăzute în majoritatea SGBD-urilor relaționale sau obiect-relaționale, ca *extensii procedurale* ale limbajului SQL
 - Oracle PL/SQL a fost dezvoltat ca extensie a limbajului SQL încă din primele implementări în SGBD Oracle; apoi a fost adaptat să fie în conformitate cu standardele obiect-relaționale (SQL:2008)
 - La fel, PostgreSQL, Transact-SQL (Microsoft SQL Server), MySQL - similare cu SQL-PSM

Standardul SQL:2008 (3)

- SQL- PSM specifică construcții care permit scrierea codului procedurilor și funcțiilor → extensii procedurale
- Instrucțiuni compuse:
BEGIN
 <statement list>
END;
- Declararea variabilelor locale:
DECLARE <data type> identifier [DEFAULT <value>] ;
- Instrucțiuni condiționale:
IF <condition> THEN <statement list>
 ELSEIF <condition> THEN <statement list>
 ELSEIF <condition> THEN <statement list>
 ELSE <statement list>
END IF;
- Instrucțiuni de ciclare (bucle):

WHILE <condition> DO <statement list> END WHILE ;	REPEAT <statement list> UNTIL <condition> END REPEAT;
---	--

Oracle SQL și PL/SQL (1)

- Oracle (versiunile actuale 11gR2, 12c) oferă cele mai avansate caracteristici obiect-relaționale și va fi folosit pentru majoritatea exemplificărilor
- Pentru dezvoltarea aplicațiilor Oracle se folosesc SQL și PL-SQL
 - SQL este limbajul de definire, modificare și interogare a bazelor de date
 - PL/SQL este extensia procedurală a limbajului SQL, conform cu SQL:2008 PSM; este folosit pentru crearea funcțiilor, procedurilor stocate, triggerelor și a pachetelor (packages)
 - În programe PL/SQL se pot introduce majoritatea instrucțiunilor SQL
- Instrucțiuni Oracle SQL (statements):
 - Data Definition Language (DDL): CREATE..., ALTER..., DROP..., GRANT
 - Data Manipulation Language (DML): CALL <procedure_name> INSERT, DELETE, UPDATE, SELECT
 - Transaction Control: SET TRANSACTION, COMMIT, ROLLBACK
 - Session Control Statements
 - System Control Statement
 - Embedded SQL Statements

Oracle SQL și PL/SQL (2)

- Unitatea de bază a unui program *PL/SQL* este *blocul*, compus din trei părți: declarații, partea executabilă, rutine de tratare a excepțiilor

[DECLARE variabile] -- opțională

BEGIN

-- partea executabila, obligatorie pentru orice bloc

[EXCEPTION rutine_tratare_exceptii] -- opțională

END;

- În partea executabilă a unui bloc (între BEGIN și END)

- Se admit:

- instrucțiuni SQL de manipulare a datelor SQL
- instrucțiuni SQL de control a tranzacțiilor
- instrucțiuni de control PL/SQL

- **NU** se admit:

- instrucțiuni SQL de definire a datelor

- Blocurile PL/SQL pot fi imbricate

- Blocurile PL/SQL pot fi:

- Blocuri anonime (care se execută imediat)
- Blocuri memorate (proceduri stocate, triggeri și funcții definite de utilizator): se compilează, se memorează și pot fi apelate apoi din orice program

Oracle SQL și PL/SQL (3)

- Instrucțiuni PL/SQL de control a execuției:
 - IF conditie THEN ... [ELSE ...] END IF;
 - CASE
 - FOR conditie LOOP
 - WHILE conditie LOOP
 - LOOP ... EXIT WHEN conditie
- Instrucț. SQL de manipulare a datelor **SELECT, INSERT, UPDATE, DELETE**:
 - Pot fi folosite interactiv, ca instrucțiuni SQL transmise SGBD-ului
 - Pot fi incluse în blocuri PL/SQL și folosite în combinație cu variabilele locale ale programului; de ex., SELECT se poate folosi astfel într-un bloc PL/SQL:
SELECT lista_coloane INTO lista_variabile
FROM lista_tabele [WHERE conditie] [optiuni];
- Instrucțiunile de definire a datelor (**CREATE, ALTER, DROP**) pot fi folosite interactiv, ca instrucțiuni SQL transmise SGBD-ului
 - Crearea tipurilor UDT, a colecțiilor de date și a subtabelelor (caract. obiect-relaționale) se face prin instrucțiunea CREATE TYPE
- Crearea și utilizarea obiectelor se poate face și în SQL și în PL/SQL

Exemplu de program PL/SQL

- În acest exemplu se crează un tabel în care se memorează rădăcinile pătrate (sq_root), pătratele (sqr) și suma pătratelor nr până la 100
- Tabelul se creează cu instrucțiunea SQL:

```
CREATE TABLE sqr_root_sum (  
    num NUMBER,  
    sq_root NUMBER(6,2),  
    sqr NUMBER,  
    sum_sqr NUMBER );  
/
```

- Inserarea valorilor în tabel se face cu blocul PL/SQL următor:

```
DECLARE  
    sum INTEGER;  
BEGIN  
    FOR i in 1..100 LOOP  
        sum := (i * (i + 1) * (2*i + 1)) / 6; -- sum of squares  
        INSERT INTO sqr_root_sum VALUES (i, SQRT(i), i*i, sum );  
    END LOOP;  
END;  
/
```

- Caracterul / este comandă de execuție a instrucțiunii SQL sau a unui bloc PL/SQL

Tipuri de date SQL

- SQL-92: tipurile de date ale atributelor sunt tipuri simple, predefinite:
 - numere (integer, floating point, numeric, decimal)
 - șiruri de caractere și de biți (de lungime fixă sau variabilă) (char, varchar etc.)
 - dată-timp (date, time, datetime, interval)
 - pentru acestea sunt prevăzute operații prin operatori și funcții SQL predefinite
- În standardul SQL:2008 (modelul OR) există:
 - Tipuri de date predefinite (built-in) – la fel ca în SQL-92
 - Tipuri de date definite de utilizator (*User-Defined Type* – *UDT*)
 - Tipuri de date furnizate de SGBD – sunt tot tipuri UDT, dar furnizate de SGBD
- Tipuri de date *UDT* furnizate de SGBD Oracle:
 - XML_types: XMLType, URIType
 - spatial_types: SDO_Geometry, SDO_Topo_Geometry, SDO_Raster
 - media_types: ORDAudio, ORDImage, ORDVideo, ORDDoc, ORDDicom
- Instrucțiunea SQL:2008 pentru crearea unui tip de date (*UDT*):
CREATE TYPE <type-name> [AS] (
 list of component attributes with individual types
 declaration of EQUAL and LESS THAN functions
 declaration of other functions (methods)
);

Definirea tipurilor de date (UDT) Oracle SQL

- Un tip de date (tip de obiecte - “*object type*”) este o clasă care conține attribute și metode:
 - Attributele memorează datele obiectului; ele pot fi valori de tipuri predefinite sau de alte tipuri UDT, vectori de valori, tabele imbricate, referințe
 - Metodele sunt proceduri sau funcții pe care se pot executa și definesc comportarea obiectelor tipului respectiv – CREATE TYPE BODY

- Exemplu Oracle SQL:

```
CREATE TYPE person_typ AS OBJECT (  
    idno NUMBER,  
    first_name VARCHAR2(20),  
    last_name VARCHAR2(25),  
    email VARCHAR2(25),  
    phone VARCHAR2(20),  
    MAP MEMBER FUNCTION get_idno RETURN NUMBER);  
/  
  
CREATE TYPE BODY person_typ AS  
    MAP MEMBER FUNCTION get_idno RETURN NUMBER IS  
        -- Metoda este definita in limbajul PL/SQL  
    BEGIN  
        RETURN idno;  
    END;  
END;  
/
```

Metodele tipurilor de date UDT (1)

- Metodele definite în tipurile UDT sunt subprograme (proceduri sau funcții) care definesc comportarea obiectelor de acel tip
- Aceste subprograme pot fi scrise în diferite limbaje și pot fi:
 - cu memorare internă (în baza de date)
 - cu memorare externă (în afara bazei de date)
- În Oracle, metodele scrise în PL/SQL și Java sunt memorate în baza de date; cele scrise în C, C++, C# sunt memorate extern
- Parametrii metodelor pot fi de intrare (IN), de ieșire (OUT), de intrare-ieșire (IN OUT); implicit sunt IN
- Metodele unui tip de date (*object type*, *class*): metode membre nestatice (ale obiectelor), metode statice (ale clasei), constructori
- Metodele membre nestatice ale tipurilor de date:
 - permit accesul la obiectele instanțe ale clasei
 - conțin un parametru predefinit (SELF) care referă obiectul instanță pentru care se invocă metoda (asemănător cu *this* din C++, Java)
 - parametrul SELF poate fi declarat explicit, dar nu este neapărat necesar

Metodele tipurilor de date UDT (2)

- Exemplu (Oracle) (NOCOPY – obiectele nu se copiază local în funcție):

```
CREATE OR REPLACE TYPE solid_typ AS OBJECT (  
    len INTEGER,  
    wth INTEGER,  
    hgt INTEGER,  
    MEMBER FUNCTION volume RETURN INTEGER,  
    MEMBER PROCEDURE display (SELF IN OUT NOCOPY solid_typ) );  
  
/  
  
CREATE OR REPLACE TYPE BODY solid_typ AS  
    MEMBER FUNCTION volume RETURN INTEGER IS  
    BEGIN  
        RETURN len * wth * hgt;  
        -- RETURN SELF.len * SELF.wth * SELF.hgt; -- equivalent to previous line  
    END;  
    MEMBER PROCEDURE display (SELF IN OUT NOCOPY solid_typ) IS  
    BEGIN  
        DBMS_OUTPUT.PUT_LINE('L: ' || len || ' - ' || 'W: ' || wth || ' - ' || 'H: ' || hgt);  
    END;  
END;  
/
```

Metode constructori

- O metodă constructor este o funcție care returnează o nouă instanță a unui tip de date; are același nume ca și tipul de date respectiv
- Invocarea constructorului se face prin numele constructorului (care este și numele tipului de date) urmat de lista de argumente; ex:
`person_typ (1, 'John', 'Smith', 'john@yahoo.com', '1-650-555-0135')`
- Pentru invocarea unui constructor se poate folosi și operatorul *new*, dar acesta nu este neapărat necesar (PL/SQL: `p := new person_typ(100, 'Ian', ...)`);
- Tipuri de constructori: generați de sistem sau definiți de utilizator
- Sistemul SGBD generează un constructor implicit pentru orice tip UDT care conține attribute (se numește *attribute-value constructor*)
- Constructorii definiți de utilizator:
 - Orice constructor definit de utilizator ascunde constructorul generat de sistem
 - Se pot defini mai mulți constructori, ca metode supraîncărcate (overloaded); selecția constructorului se face după numărul sau tipul argumentelor
 - Au ca prim parametru obligatoriu SELF (declarat explicit sau nu)
 - Returnează obligatoriu SELF: `RETURN SELF AS RESULT`

Exemplu de constructori definiți de utilizator

```
CREATE TYPE shape AS OBJECT (  
    name VARCHAR2(30),  
    area NUMBER,  
    CONSTRUCTOR FUNCTION shape (SELF IN OUT NOCOPY shape, name VARCHAR2)  
        RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION shape (SELF IN OUT NOCOPY shape, name VARCHAR2,  
        area NUMBER) RETURN SELF AS RESULT  
)  
NOT FINAL;  
/  
  
CREATE TYPE BODY shape AS  
    CONSTRUCTOR FUNCTION shape (SELF IN OUT NOCOPY shape, name VARCHAR2)  
        RETURN SELF AS RESULT IS  
    BEGIN  
        SELF.name := name;  
        area := 0;  
        RETURN;  
    END;  
    CONSTRUCTOR FUNCTION shape (SELF IN OUT NOCOPY shape, name VARCHAR2,  
        area NUMBER) RETURN SELF AS RESULT IS  
    BEGIN  
        SELF.name := name;  
        SELF.area := area;  
        RETURN;  
    END;  
END;  
/
```

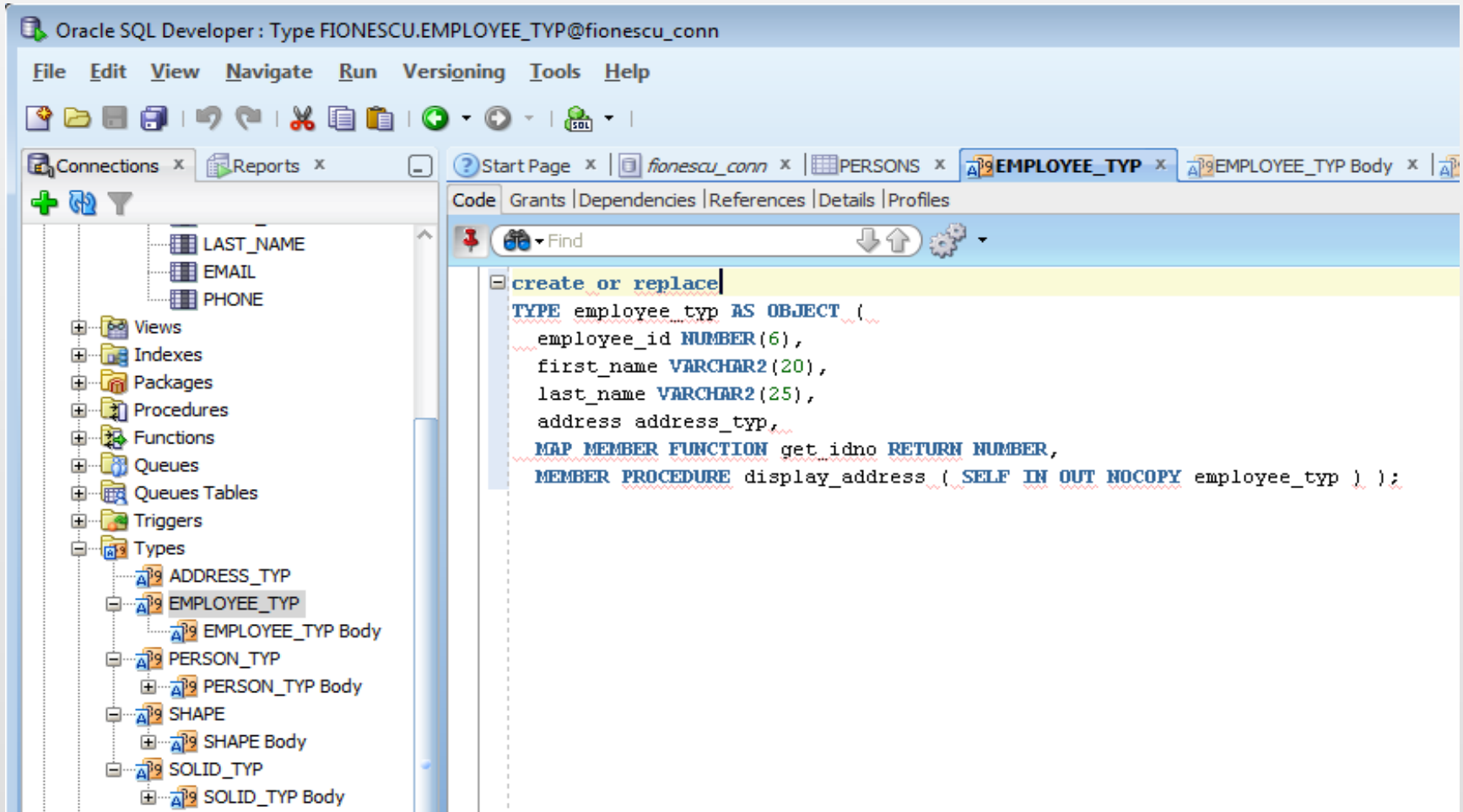
Exemple de creare a tipurilor de date (1)

- Se creează *object types* cu instrucțiunea SQL CREATE TYPE; tipul *employee_typ* are attribute de tipuri predefinite (*varchar2*) și un atribut de tip UDT (*address_typ*)

```
CREATE TYPE address_typ AS OBJECT (  
    street VARCHAR2(30),  
    city VARCHAR2(20),  
    numar INTEGER);  
/  
CREATE TYPE employee_typ AS OBJECT (  
    employee_id NUMBER(6),  
    first_name VARCHAR2(20),  
    last_name VARCHAR2(25),  
    address address_typ,  
    MAP MEMBER FUNCTION get_idno RETURN NUMBER,  
    MEMBER PROCEDURE display_address ( SELF IN OUT NOCOPY employee_typ ) );  
/  
CREATE TYPE BODY employee_typ AS  
    MAP MEMBER FUNCTION get_idno RETURN NUMBER IS  
    BEGIN  
        RETURN employee_id;  
    END;  
    MEMBER PROCEDURE display_address ( SELF IN OUT NOCOPY employee_typ ) IS  
    BEGIN  
        DBMS_OUTPUT.PUT_LINE(first_name || ' ' || last_name);  
        DBMS_OUTPUT.PUT_LINE(address.street || ', ' || address.city || ' ' || address.numar);  
    END;  
END;  
/
```

Exemple de creare a tipurilor de date (2)

- În SQL Developer se pot inspecta tipurile de date create (definiția atributelor și a metodelor)
 - Obs. Codul de creare este rescris de toolset, în format standard



Creare și accesarea obiectelor în blocuri PL/SQL

- Se pot crea obiecte (instanțe ale unor tipuri de date) în blocuri PL/SQL
- Pentru accesul la metodele și atributele obiectelor se folosește operatorul de selecție membru (.)

```
DECLARE
```

```
    emp employee_typ; -- declare object emp; it is atomically null
```

```
BEGIN
```

```
    -- call the constructor for employee_typ; new is optional
```

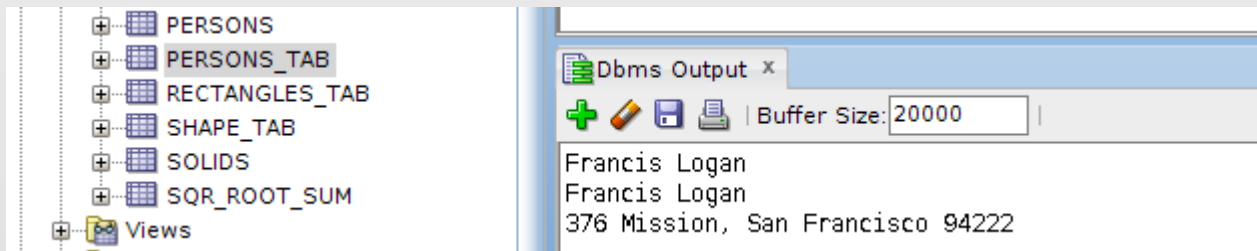
```
    emp := new employee_typ(315, 'Francis', 'Logan',  
    address_typ('376 Mission', 'San Francisco', 94222));
```

```
    DBMS_OUTPUT.PUT_LINE(emp.first_name || ' ' || emp.last_name); -- display details
```

```
    emp.display_address(); -- call object method to display details
```

```
END;
```

```
/
```

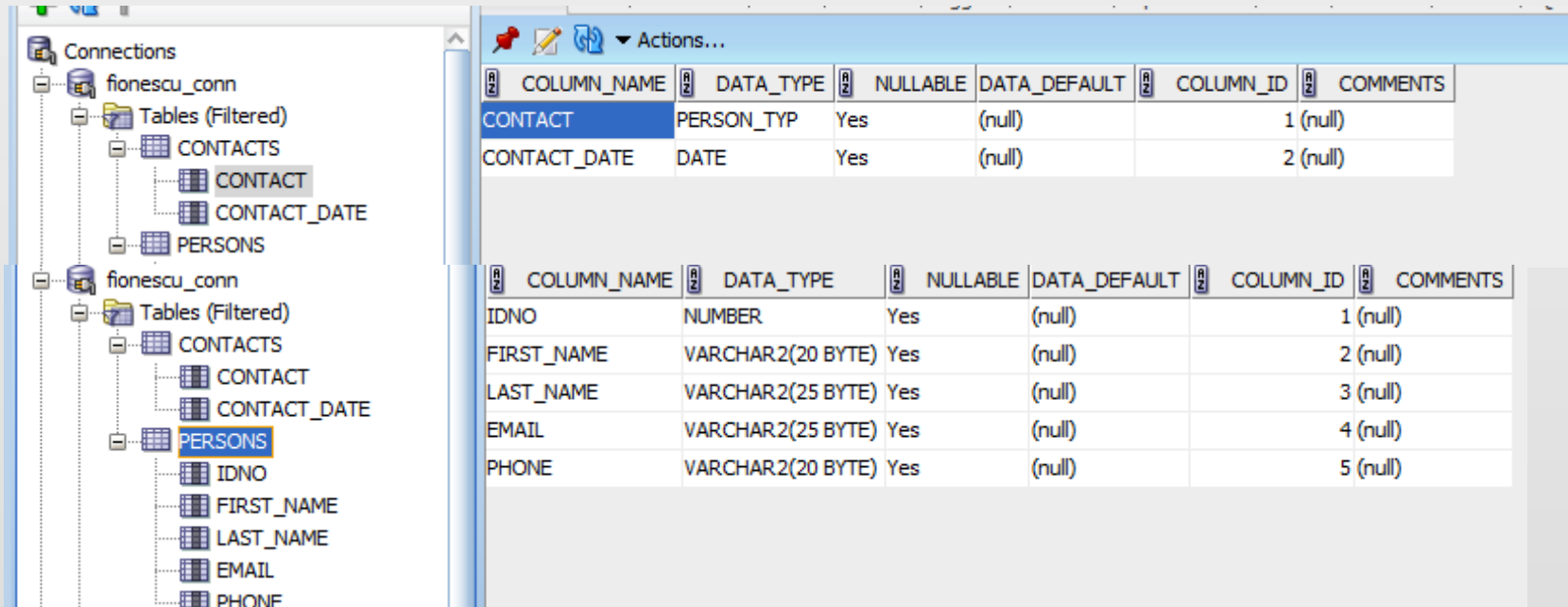


Crearea tabelelor

- În sistemele Oracle se pot crea două categorii de tabele: tabele relaționale și tabele de obiecte
- **Tabele relaționale** (*relational table*) – se crează cu instrucțiunea CREATE TABLE și poate conține mai multe atribute (coloane); tipul unui atribut poate fi: tip predefinit, tip UDT, vector de date, tabel imbricat
 - Valoarea unui atribut (coloană) de tip UDT este un obiect instanță a acelui tip; se mai numește și obiect coloană (*column object*)
 - De exemplu, tabelul “*contacts*” este definit ca tabel relațional, cu o coloană (*contact_date*) de tip predefinit și o coloană (*contact*) de tipul UDT *person_typ*;
CREATE TABLE contacts (
 contact *person_typ*,
 contact_date DATE);
- **Tabele de obiecte** (*object table*): fiecare linie conține un obiect de un tip UDT (numit obiect linie – *row object*) ; de exemplu:
CREATE TABLE persons OF person_typ;
- Un astfel de tabel poate fi privit ca:
 - Un tabel cu o singură coloană care conține numai obiecte (*object table*); fiecare linie conține un obiect de tipul dat (*row object*)
 - Un tabel cu coloanele corespunzătoare atributelor tipului dat

Exemplu: crearea tabelelor

- In SQL Developer se observă că:
 - CONTACTS este un *tabel relational* care conține:
 - un atribut (CONTACT) care este de tipul UDT PERSON_TYP
 - un atribut (CONTACT_DATE) care este de tip SQL predefinit (DATE)
 - PERSONS este un *tabel de obiecte* compus dintr-o coloană de tip UDT și fiecare linie conține un obiect de acel tip (PERSON_TYP)
 - SQL Developer reprezintă un tabel de obiecte ca tabel care are ca și coloane attributele tipului respectiv



The screenshot shows the SQL Developer interface. On the left, the 'Connections' pane displays two connections to 'fionescu_conn'. The first connection shows a 'Tables (Filtered)' folder containing 'CONTACTS' and 'PERSONS'. The second connection shows a 'Tables (Filtered)' folder containing 'CONTACTS', 'CONTACT', 'CONTACT_DATE', and 'PERSONS'. The 'PERSONS' table is selected. On the right, the 'Actions...' pane displays the table structure for 'PERSONS'.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
CONTACT	PERSON_TYP	Yes	(null)	1 (null)	
CONTACT_DATE	DATE	Yes	(null)	2 (null)	

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
IDNO	NUMBER	Yes	(null)	1 (null)	
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2 (null)	
LAST_NAME	VARCHAR2(25 BYTE)	Yes	(null)	3 (null)	
EMAIL	VARCHAR2(25 BYTE)	Yes	(null)	4 (null)	
PHONE	VARCHAR2(20 BYTE)	Yes	(null)	5 (null)	

Cheile primare ale tabelelor

- Pentru tabelele relaționale cheile primare se definesc printr-un atribut (simplu sau compus) al relației - la fel ca în modelul relațional
- Pentru tabelele de obiecte cheile primare se definesc printr-un atribut (simplu sau compus) al tipului de obiecte
- Exemplu:

```
-- CREATE OR REPLACE TYPE person_typ AS OBJECT (...) a fost creat deja
```

```
CREATE OR REPLACE TYPE location_typ AS OBJECT (
```

```
    building_no NUMBER,
```

```
    city VARCHAR2(40) );
```

```
/
```

```
CREATE OR REPLACE TYPE office_typ AS OBJECT (
```

```
    office_id  VARCHAR(10),
```

```
    office_loc location_typ,
```

```
    occupant  person_typ );
```

```
/
```

```
CREATE TABLE office_tab OF office_typ (
```

```
    office_id  PRIMARY KEY );
```

```
/
```

Crearea, inițializarea și accesarea obiectelor coloană

- Se pot crea obiecte coloană în SQL, prin instrucțiuni INSERT; la crearea obiectelor se invocă constructorii care le și inițializează

```
CREATE TABLE persons_tab (
```

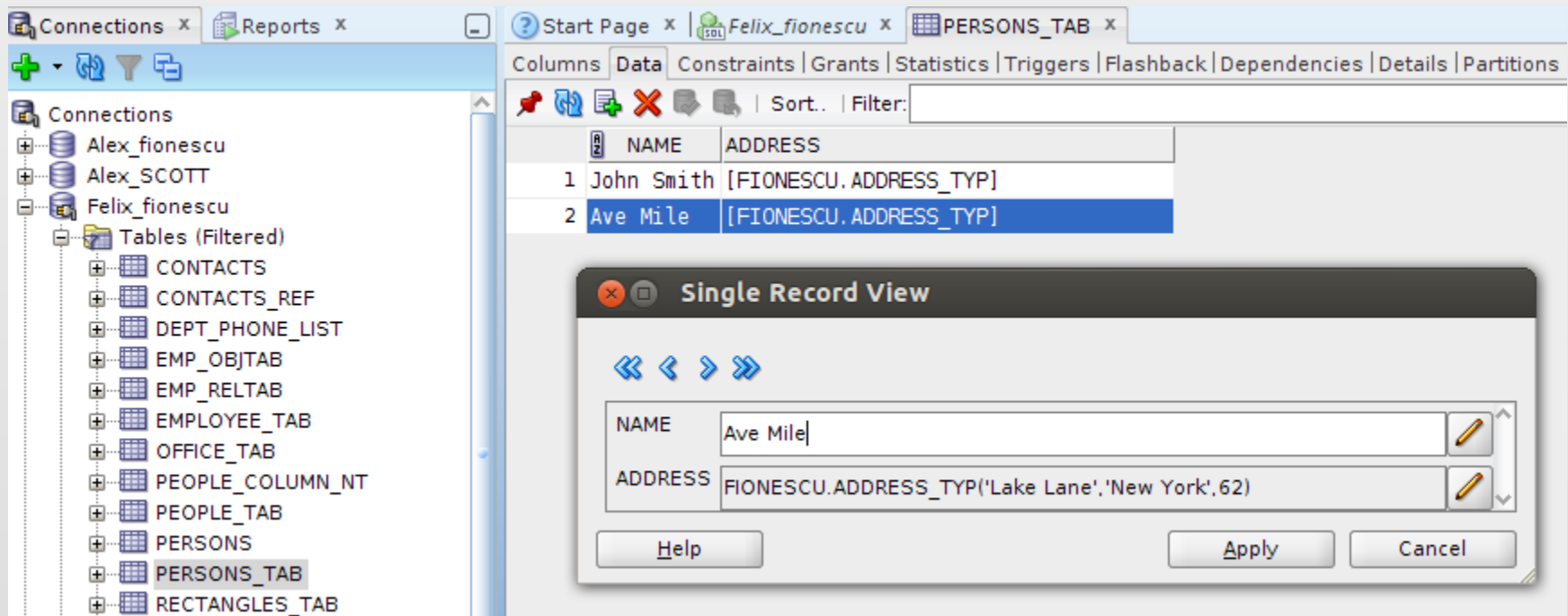
```
    name varchar2(20),
```

```
    address address_typ);
```

```
/
```

```
INSERT INTO persons_tab VALUES ('John Smith', address_typ ('Rock', 'New York', 17) );
```

```
INSERT INTO persons_tab VALUES('Ave Mile', address_typ ('Lake Lane', 'New York', 62) );
```



Utilizarea tabelelor de obiecte în SQL

- Crearea tabelelor de obiecte:

CREATE TABLE solids of solid_typ;

- Inserarea obiectelor în tabel – se apelează implicit sau explicit constructorul tipului UDT iar datele se introduc ca valori ale atributelor tipului de obiecte:

INSERT INTO solids VALUES(10, 10, 10);

-- SAU: INSERT INTO solids VALUES(**solid_typ**(10,10,10));

INSERT INTO solids VALUES(3, 4, 5);

- Interogare folosind limbajul SQL:

- se definește o variabilă tabel (table alias - solids s, în exemplul dat)

- se folosește operatorul de selecție membru (.) pentru acea variabilă
select s.volume() from solids s where s.len=10;

The screenshot displays a SQL IDE interface. On the left, a tree view shows the database structure under 'fionescu_conn', including tables like CONTACTS, EMPLOYEE_TAB, PERSONS, PERSONS_TAB, and SOLIDS. The main window shows the following SQL script:

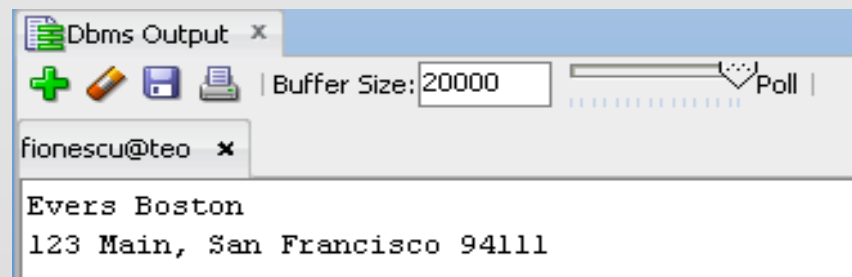
```
CREATE TABLE solids of solid_typ;
INSERT INTO solids VALUES(10, 10, 10);
INSERT INTO solids VALUES(3, 4, 5);
SELECT * FROM solids;
SELECT s.volume() FROM solids s WHERE s.len = 10;
SELECT len from solids;
```

Below the script, the 'Query...' tab is active, showing the results of the last query. The status bar indicates 'All Rows Fetched: 2 in 0.003 seconds'.

	LEN
1	10
2	3

Utilizarea tabelelor de obiecte în blocuri PL/SQL

- Fie tabelul employee_tab creat cu comanda SQL:
CREATE TABLE employee_tab of employee_typ;
/
/
- În blocul PL/SQL următor se creează și se citesc obiecte (linii) în tabel:
DECLARE emp employee_typ;
BEGIN
 INSERT INTO employee_tab VALUES (employee_typ(310, 'Evers', 'Boston',
 address_typ('123 Main', 'San Francisco', 94111)));
 INSERT INTO employee_tab VALUES (employee_typ(321, 'Martha', 'Dunn',
 address_typ('123 Broadway', 'Redwood City', 94065)));
 SELECT VALUE(e) INTO emp FROM employee_tab e WHERE e.employee_id = 310;
 emp.display_address();
END;
/
/
- Funcția VALUE are ca argument o variabilă tabel de obiecte (object table alias) și returnează obiectele corespunzătoare liniilor tabelului care îndeplinesc condiția dată



Metode de comparare a obiectelor

- Valorile tipurilor scalare predefinite (char, float, numeric etc.) pot fi comparate și ordonate, deoarece au o ordine predefinită
- Obiectele conțin valori ale atributelor proprii de diferite tipuri și nu pot fi comparate sau ordonate direct
- Pentru obiecte este necesar definirea unei metode de mapare (MAP) **SAU** a unei metode de ordonare (ORDER); **NU** ambele!
- Funcția de comparare definită este invocată automat atunci când se compară două obiecte de acel tip (în clauzele WHERE, DISTINCT, GROUP BY, ORDER BY)
- Metoda de mapare (MAP MEMBER FUNCTION):
 - returnează o valoare de un tip scalar predefinit, calculată din valorile atributelor obiectului
 - comparația `obj_1 > obj_2` este echivalentă cu: `obj_1.map() > obj_2.map()`
- Metoda de ordonare (ORDER MEMBER FUNCTION) compară obiectul curent cu obiectul dat ca argument pe baza unor anumite criterii și returnează:
 - valoare negativă (dacă obiectul curent este mai mic decât obiectul argument),
 - 0 (obiectele sunt egale)
 - valoare pozitivă (dacă obiectul curent este mai mare decât obiectul argument)

Metode de mapare și ordonare

- Exemplu: se definește metoda *area* pentru comparația a două obiecte de tipul `rectangle_typ`:

```
CREATE OR REPLACE TYPE rectangle_typ AS OBJECT (  
    len NUMBER,  
    wid NUMBER,  
    -- se poate defini o metoda de ordonare in locul metodei de mapare ; NU ambele  
    -- MAP MEMBER FUNCTION area RETURN NUMBER);  
ORDER MEMBER FUNCTION area (r rectangle_typ) RETURN INTEGER );  
/  
CREATE OR REPLACE TYPE BODY rectangle_typ AS  
    /* MAP MEMBER FUNCTION area RETURN NUMBER IS  
    BEGIN  
        RETURN len * wid;  
    END; */  
ORDER MEMBER FUNCTION area (r rectangle_typ) RETURN INTEGER IS  
    BEGIN  
        IF      len*wid < r.len*r.wid THEN RETURN -1; -- any negative number will do  
        ELSIF len*wid > r.len*r.wid THEN RETURN 1; -- any positive number will do  
        ELSE RETURN 0;  
    END IF;  
    END;  
END;  
/
```

Exemplu de utilizare a metodei de ordonare

- Se definește tabelul RECTANGLES_TAB:

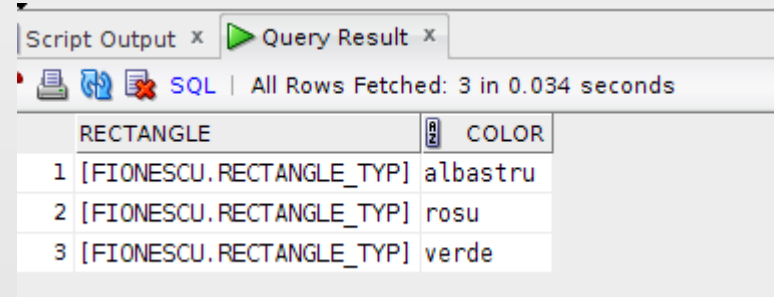
```
CREATE TABLE RECTANGLES_TAB(  
  rectangle RECTANGLE_TYP,  
  color varchar2(10));  
/
```

- Se inserează valori în tabel:

```
INSERT INTO RECTANGLES_TAB VALUES (rectangle_typ(2,4), 'rosu');  
INSERT INTO RECTANGLES_TAB VALUES (rectangle_typ(7,3), 'verde');  
INSERT INTO RECTANGLES_TAB VALUES (rectangle_typ(5,9), 'albastru');
```

- Ordonare linii după atributul *color*:

```
SELECT * FROM  
  RECTANGLES_TAB  
  order BY color;
```



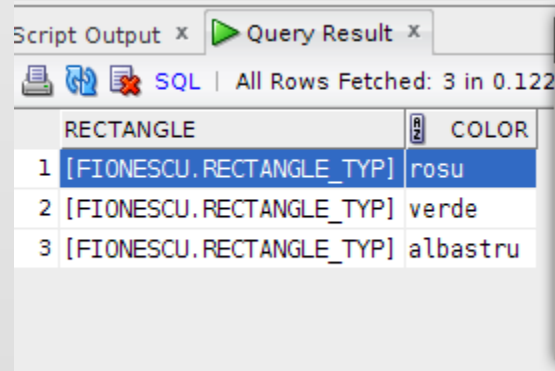
Script Output x Query Result x

SQL | All Rows Fetched: 3 in 0.034 seconds

	RECTANGLE	COLOR
1	[FIONESCU.RECTANGLE_TYP]	albastru
2	[FIONESCU.RECTANGLE_TYP]	rosu
3	[FIONESCU.RECTANGLE_TYP]	verde

- Ordonare linii după atributul rectangle - ordonarea după area():

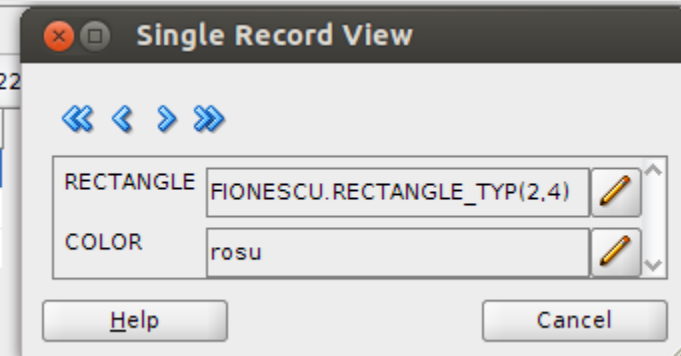
```
SELECT * FROM  
  RECTANGLES_TAB  
  order BY rectangle;
```



Script Output x Query Result x

SQL | All Rows Fetched: 3 in 0.122

	RECTANGLE	COLOR
1	[FIONESCU.RECTANGLE_TYP]	rosu
2	[FIONESCU.RECTANGLE_TYP]	verde
3	[FIONESCU.RECTANGLE_TYP]	albastru



Single Record View

Navigation: << < > >>

RECTANGLE: FIONESCU.RECTANGLE_TYP(2,4)

COLOR: rosu

Buttons: Help, Cancel

Metode statice

- Metodele statice sunt invocate pentru un tip de date UDT, nu pentru instanțe ale acestuia
- Metodele statice nu au parametrul SELF
- Metodele statice se declară folosind specificatorii: **STATIC FUNCTION** sau **STATIC PROCEDURE**.
- Apelul metodelor statice se face calificând metoda (folosind operatorul de selecție membru **.**) cu numele tipului, nu cu numele unui obiect instanță al acestuia
- De exemplu:
`type_name.method()`

Identificatorii obiectelor și referințe la obiecte

- **Obiectele linie** (“*row objects*”) dintr-un tabel de obiecte pot fi identificate prin identificatori (OIDs – Object IDentifiers), care pot fi de două feluri:
 - OID generați de sistem
 - OID bazați pe chei primare – definite de proiectant
- OID-urile identifică obiecte persistente (în baza de date), nu obiecte în memorie (nepersistente)
- **Obiectele coloană** (“*column objects*”) dintr-un tabel relațional sunt identificate prin cheia primară a liniei (tuplului) din care face parte și nu necesită OID-uri
- O referință REF T (unde T este un tip de date) este un “pointer” logic care permite identificarea unui obiect linie de tipul T folosind OID-ul
- O referință poate fi modificată astfel încât să indice un obiect diferit, dar de același tip T (sau un supertip al acestuia)
- Referințele modelează asocierile N:1, și pot înlocui cheile străine:

```
CREATE TYPE emp_typ AS OBJECT (  
    emp_id number,  
    name VARCHAR2(30),  
    manager REF emp_typ );  
/  
CREATE TABLE emp_objtab OF emp_typ;  
/
```

Referințe la obiecte

- Se introduc date in tabelul emp_objtab:

```
INSERT INTO emp_objtab VALUES (1,'Evens Brown', NULL);
```

```
INSERT INTO emp_objtab VALUES (2, 'Martin Joe', (SELECT REF(e)  
FROM emp_objtab e WHERE e.emp_id = 1));
```

- Se afișează conținutul tabelului:

```
SELECT * FROM emp_objtab;
```

The screenshot displays a database management interface. On the left, a tree view shows the database structure, including tables like CONTACTS, CONTACTS_REF, DEPT_PHONE_LIST, EMP_OBJTAB, and EMP_RELTAB. The main window shows a SQL query: `SELECT * FROM emp_objtab;` with a 'Query Result' tab active. The query result is a table with three columns: EMP_ID, NAME, and MANAGER. It contains two rows: (1, 'Evens Brown', (null)) and (2, 'Martin Joe', [FIONESCU.EMP_TYP]). A 'Single Record View' dialog box is open, showing the details of the second row. The dialog has fields for EMP_ID (2), NAME (Martin Joe), and MANAGER (FIONESCU.EMP_TYP(1,'Evens Brown',NULL)). The dialog also includes navigation buttons (back, forward, first, last) and 'Help' and 'Cancel' buttons.

EMP_ID	NAME	MANAGER
1	Evens Brown	(null)
2	Martin Joe	[FIONESCU.EMP_TYP]

Referințele la obiecte pot înlocui cheile străine

- Varianta fără referințe a tabelului emp_objtab poate fi scrisă astfel:

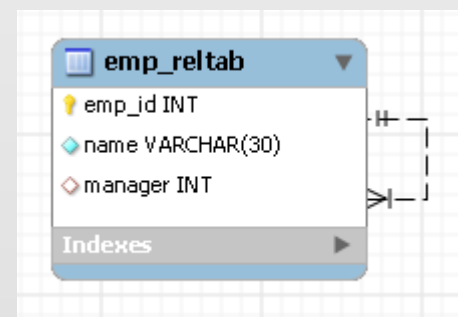
```
CREATE TABLE emp_reltab (  
  emp_id number PRIMARY KEY,  
  name varchar2(30),  
  manager number,  
  FOREIGN KEY (manager) REFERENCES emp_reltab(emp_id));  
/
```

- Se introduc date în tabelul emp_reltab:

```
INSERT INTO emp_reltab VALUES (1,'Evens Brown', NULL);  
INSERT INTO emp_reltab VALUES (2, 'Martin Joe', (SELECT e.emp_id  
FROM emp_reltab e WHERE e.name = 'Evens Brown'));
```

```
CREATE TABLE emp_reltab (  
  emp_id number PRIMARY KEY,  
  name varchar2(30),  
  manager number,  
  CONSTRAINT FK1 FOREIGN KEY (manager) REFERENCES emp_reltab(emp_id));  
/  
INSERT INTO emp_reltab VALUES(1,'Evens Brown', NULL);  
INSERT INTO emp_reltab VALUES(2, 'Martin Joe', (SELECT e.emp_id  
  FROM emp_reltab e WHERE e.name = 'Evens Brown'));  
select * FROM emp_reltab;
```

Script Output x Statement Output x Query Result x			
All Rows Fetched: 2 in 0 seconds			
EMP_ID	NAME	MANAGER	
1	1 Evens Brown	(null)	
2	2 Martin Joe	1	



Constrângeri asupra referințelor (1)

■ Referințe:

- fără constrângeri – o referință poate indica un obiect de tipul referit din orice tabel, sau poate să nu indice nici-un obiect existent (“*dangling references*”)
- cu constrângeri de domeniu (SCOPE) - obligă referința să indice numai obiecte din tabelul referit

■ Exemplu:

```
CREATE TYPE person_typ AS OBJECT ( -- acest tip a fost deja definit  
    idno NUMBER,.....);
```

```
CREATE TABLE persons OF person_typ;
```

```
CREATE TABLE contacts_ref (  
    contact_ref REF person_typ SCOPE IS persons,  
    contact_date DATE );  
/
```

```
INSERT INTO PERSONS VALUES (1, 'Evens', 'Brown','ebrown@yahoo.com','02134567');
```

```
INSERT INTO CONTACTS_REF VALUES ((SELECT REF(p) FROM PERSONS p  
    WHERE p.last_name='Brown'),TO_DATE('20-MAR-2013', 'DD-MON-YYYY'));
```

```
SELECT * FROM CONTACTS_REF;
```

```
/
```

Constrângeri asupra referințelor (2)

- În SQL Developer se afisează:

The screenshot displays the SQL Developer interface. The main window shows a SQL script with the following content:

```
CREATE TABLE contacts_ref (  
  contact_ref REF person_typ SCOPE IS persons,  
  contact_date DATE );  
/  
INSERT INTO PERSONS VALUES(1, 'Evens', 'Brown', 'ebrown@yahoo.com', '02134567');  
INSERT INTO CONTACTS_REF VALUES((SELECT REF(p) FROM PERSONS p  
  WHERE p.last_name='Brown'),TO_DATE('20-MAR-2013', 'DD-MON-YYYY'));  
SELECT * FROM CONTACT_REFS;
```

Below the script, the 'Query Result' tab is active, showing a table with two columns: 'CONTACT_REF' and 'CONTACT_DATE'. The first row contains the values '1' and '20-MAR-13'.

A 'Single Record View' dialog box is overlaid on the query result. It contains the following fields:

- CONTACT_REF: FIONESCU.PERSON_TYP(1,'Evens','Brown','ebrown@yahoo.com','02134567')
- CONTACT_DATE: 20-MAR-13

The dialog box also includes a 'Help' button and a 'Cancel' button.

Crearea referințelor și dereferențierea

- O referință la un obiect linie se poate obține prin selectarea obiectului din tabelul său de obiecte și aplicând operatorul REF
- Accesarea unui obiect pentru care avem referința acestuia se numește dereferențiere
- Oracle oferă operatorul Deref pentru operația de dereferențiere
- Exemplu: obținerea referinței la obiectul persoanei care are idno 1:

```
DECLARE
    person_ref REF person_typ;
    person person_typ;
BEGIN
    SELECT REF(p) INTO person_ref
    FROM persons p
    WHERE p.idno = 1;
    SELECT Deref(person_ref) into person from dual; -- use dummy table DUAL
    DBMS_OUTPUT.PUT_LINE(person.FIRST_NAME || ' ' || person.LAST_NAME);
END;
```

- În fereastra DBMS Output se afișează Evens Brown - dacă s-au folosit val din text

Moștenirea tipurilor UDT

- Un subtip reprezintă o specializare a unui tip dat (numit *supertip*) și se definește folosind cuvântul cheie *under*:

```
CREATE TYPE student_typ UNDER person_typ (  
    department varchar2(20))  
/;
```

- Un subtip moștenește:
 - attributele definite sau moștenite de supertip
 - metodele definite sau moștenite de supertip(cu excepția constructorilor)
- Metodele moștenite pot fi redefinite (*redefined*) și declarate dominante (*override*)
- Tipurile, attributele și metodele declarate FINAL nu pot fi specializate sau redefinite
- Ierarhii de tipuri de date - supertipuri și subtipuri
- Unele sisteme (printre care și Oracle) admit numai moștenirea simplă

Exemplu - Moștenire (1)

```
CREATE TYPE rectangle UNDER shape (  
    len NUMBER, wth NUMBER,  
    CONSTRUCTOR FUNCTION rectangle(SELF IN OUT NOCOPY rectangle,  
    name VARCHAR2, len NUMBER, wth NUMBER) RETURN SELF as RESULT,  
    CONSTRUCTOR FUNCTION rectangle(SELF IN OUT NOCOPY rectangle,  
    name VARCHAR2, side NUMBER) RETURN SELF as RESULT);  
/  
CREATE TYPE BODY rectangle IS  
    CONSTRUCTOR FUNCTION rectangle (SELF IN OUT NOCOPY rectangle,  
    name VARCHAR2, len NUMBER, wth NUMBER) RETURN SELF AS RESULT IS  
    BEGIN  
        SELF.name := name;    SELF.area := len*wth;  
        SELF.len := len;      SELF.wth := wth;  
        RETURN ;  
    END;  
    CONSTRUCTOR FUNCTION rectangle (SELF IN OUT NOCOPY rectangle,  
    name VARCHAR2, side NUMBER) RETURN SELF AS RESULT IS  
    BEGIN  
        SELF.name := name; SELF.area := side * side;  
        SELF.len := side; SELF.wth := side;  
        RETURN ;  
    END;  
END;  
/
```

Exemplu - Moștenire (2)

- Într-un tabel de obiecte de un tip dat se pot insera obiecte de tipul respectiv și de subtipuri ale acestuia
- De exemplu: în tabelul de obiecte de tip *shape* se pot insera obiecte *shape* și *rectangle*:

```
CREATE TABLE shape_tab OF shape;
```

```
/
```

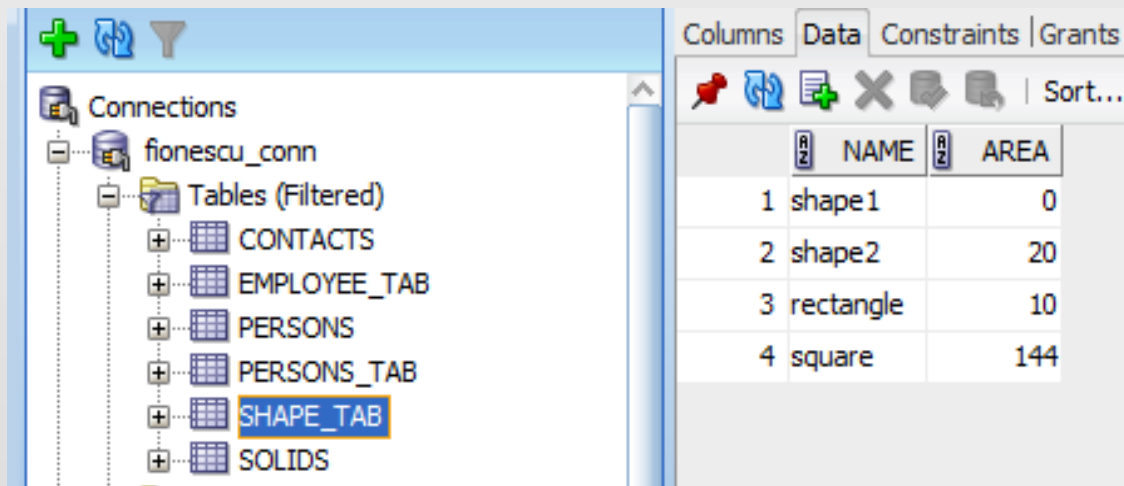
```
INSERT INTO shape_tab VALUES(shape('shape1'));
```

```
INSERT INTO shape_tab VALUES(shape('shape2', 20));
```

```
INSERT INTO shape_tab VALUES(rectangle('rectangle', 2, 5));
```

```
INSERT INTO shape_tab VALUES(rectangle('square', 12));
```

```
SELECT * FROM shape_tab;
```



The screenshot shows the Oracle SQL Developer interface. On the left, the 'Connections' pane displays a tree view with 'fionescu_conn' expanded, showing a list of tables: CONTACTS, EMPLOYEE_TAB, PERSONS, PERSONS_TAB, SHAPE_TAB (highlighted), and SOLIDS. On the right, the 'Data' tab of the 'SHAPE_TAB' table is active, displaying a table with two columns: NAME and AREA. The table contains four rows of data.

	NAME	AREA
1	shape1	0
2	shape2	20
3	rectangle	10
4	square	144

Colecții de date

- Modelul OR suportă colecții de date: vectori de date (*varray*) și tabele imbricate (*nested tables*); sunt folosite pentru attribute cu valori multiple
 - Un vector de date este o colecție ordonată de elemente de același tip
 - Un tabel îmbricat este o colecție cu un număr oarecare de elemente
- În Oracle, un vector (*varray*) este o mulțime ordonată de elemente:
 - Elementele sunt de același tip sau de un subtip al tipului declarat
 - Fiecare element are un index, care este poziția elementului în vector
 - La definirea unui vector se specifică numărul maxim de elemente admis
 - Numărul real de elemente poate varia, fără a depăși limita impusă

- Exemplu:

```
CREATE TYPE phone_typ AS OBJECT (  
    country_code VARCHAR2(2),  
    area_code VARCHAR2(3),  
    ph_number VARCHAR2(7));  
/  
CREATE TYPE phone_varray_typ AS VARRAY(5) OF phone_typ;  
/  
CREATE TABLE dept_phone_list (  
    dept_no NUMBER(5),  
    phone_list phone_varray_typ);  
/
```


Vectori de date

- Se inserează date în tabel:

```
INSERT INTO dept_phone_list VALUES (100, phone_varray_typ ( phone_typ ('01', '650',  
'5550123'), phone_typ ('01', '650', '5550148'), phone_typ ('01', '650', '5550192')));
```

- Se afișează liniile tabelului:

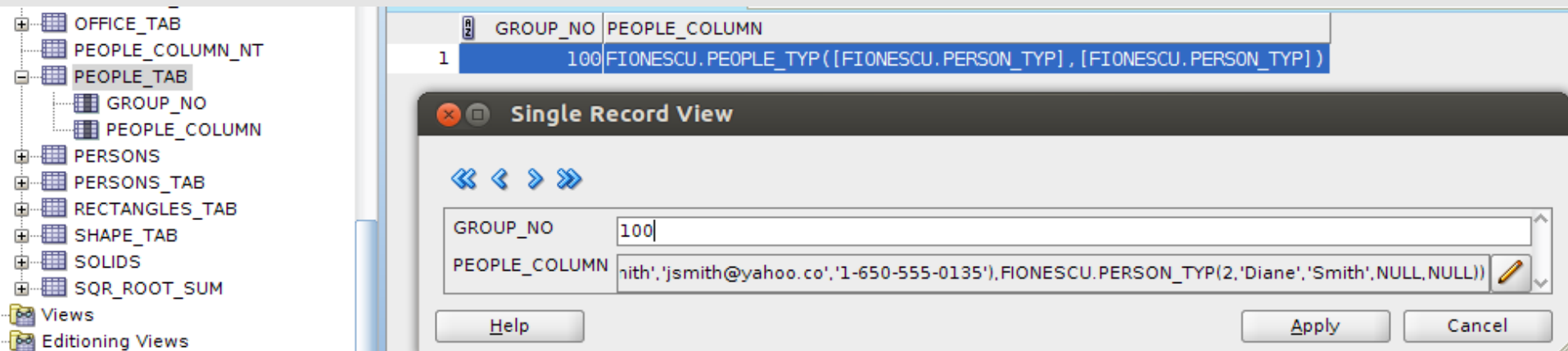
```
SELECT * FROM dept_phone_list;
```

The screenshot displays a database management interface. On the left, a tree view shows the database structure, including tables like CONTACTS, CONTACTS_REF, CONTACT_REF, CONTACT_DATE, DEPT_PHONE_LIST, DEPT_NO, PHONE_LIST, EMP_OBJTAB, EMP_RELTAB, EMPLOYEE_TAB, OFFICE_TAB, PEOPLE_COLUMN_NT, PEOPLE_TAB, PERSONS, PERSONS_TAB, RECTANGLES_TAB, SHAPE_TAB, SOLIDS, and SQR_ROOT_SUM. The main window shows the SQL query `SELECT * FROM dept_phone_list;` executed. The results are displayed in a table with two columns: DEPT_NO and PHONE_LIST. The first row shows DEPT_NO 100 and PHONE_LIST FIONESCU.PHONE_VARRAY_TYP([FIONESCU.PHONE_TYP], [FIONESCU.PHONE_TYP], [FIONESCU.PHONE_TYP]). A 'Single Record View' dialog box is open, showing the details of the first row. The DEPT_NO field is 100, and the PHONE_LIST field is TYP('01','650','5550123'),FIONESCU.PHONE_TYP('01','650','5550148'),FIONESCU.PHONE_TYP('01','650','5550192'). A 'View Value' dialog box is also open, showing the value of the PHONE_LIST field: FIONESCU.PHONE_VARRAY_TYP([FIONESCU.PHONE_TYP], [FIONESCU.PHONE_TYP], [FIONESCU.PHONE_TYP]).

DEPT_NO	PHONE_LIST
100	FIONESCU.PHONE_VARRAY_TYP([FIONESCU.PHONE_TYP], [FIONESCU.PHONE_TYP], [FIONESCU.PHONE_TYP])

Tabele imbricate

- Exemplu – reluam tipul `person_typ` si cream un tip de tabel imbricat:
CREATE TYPE `people_typ` **AS TABLE OF** `person_typ`; -- nested table type
/
- După crearea tipului unui tabel imbricat (*nested table*), se poate folosi o instanță a acestui tip ca valoare a unui atribut al altui tabel:
CREATE TABLE `people_tab` (
 `group_no` NUMBER,
 `people_column` `people_typ`) -- an instance of nested table
 NESTED TABLE `people_column` STORE AS `people_column_nt`;
/
INSERT INTO `people_tab` **VALUES** (100,
 `people_typ`(`person_typ`(1, 'John','Smith', 'jsmith@yahoo.co','1-650-555-0135'),
 `person_typ`(2, 'Diane','Smith', NULL, NULL)));
- O instanță a unui tip de colecție se creează la fel ca o instanță a oricărui tip (*object type*), prin apelul metodei constructor al acelui tip



Memorarea tabelelor imbricate

- Oracle memorează tabelele imbricate într-o zonă de memorie separată
- Un identificator NESTED_TABLE_ID generat de SGBD pe 16 biți corelează linia părinte cu liniile corespunzătoare din tabelul imbricat
- Datele imbricate sunt grupate în tabele corespunzătoare liniilor din tabelul părinte (tabelul A, tabelul B etc.)
- În coloana NT_DATA din tabelul părinte se memorează ID tabelului imbricat (A, B, C etc.)
- Identific. NESTED_TABLE_ID pot fi creați ca:
 - prefix al cheii primare a tabelului părinte
 - indexuri în tabelul imbricat (IOT – Indexed Organized Table); aceștia pot fi și compresăți (COMPRESSED)

Nested Table Storage

DATA1	DATA2	DATA3	DATA4	NT_DATA
...	A
...	B
...	C
...	D
...	E

Storage Table

NESTED_TABLE_ID	Values
A	A11
A	A12
A	A13
B	B21
B	B22
B	B23
B	B24
B	B25
C	C31
C	C32
D	D41
E	E51
E	E52
E	E53
E	E54

Storage for nested table A

Storage for nested table B

Storage for nested table C

Storage for nested table D

Storage for nested table E

Rezumat: crearea tipurilor de date si a tabelelor

- Crearea unui tip UDT:

```
CREATE TYPE person_typ AS OBJECT (  
    idno NUMBER, first_name VARCHAR2(20), last_name VARCHAR2(25),  
    email VARCHAR2(25), phone VARCHAR2(20));
```

- Crearea unui tabel de obiecte:

```
CREATE TABLE persons OF person_typ;
```

- Crearea unui tabel relational cu o coloana de tip UDT:

```
CREATE TABLE contacts (  
    contact person_typ, contact_date date);
```

- Crearea unui tip pentru vectori VARRAY

```
CREATE TYPE person_varray_typ AS VARRAY(5) OF person_typ;
```

- Crearea unui tabel relational cu o coloana varray:

```
CREATE TABLE people_varray_tab (  
    group_no number, person_varray person_varray_typ );
```

- Definirea unui tabel de obiecte ca tip pentru tabele imbricate (nested table)

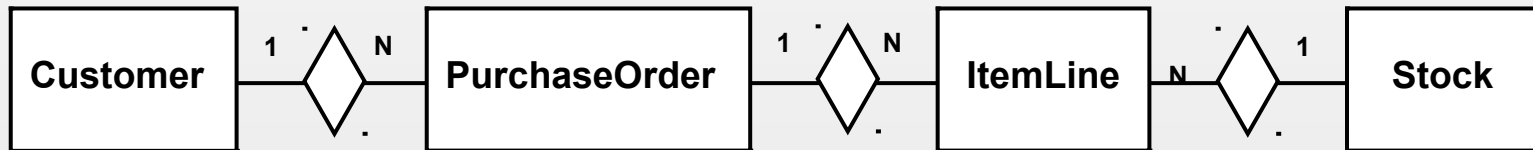
```
CREATE TYPE people_typ AS TABLE OF person_typ; -- nested table type
```

- Crearea unui tabel relational cu o coloana tabel imbricat (nested table):

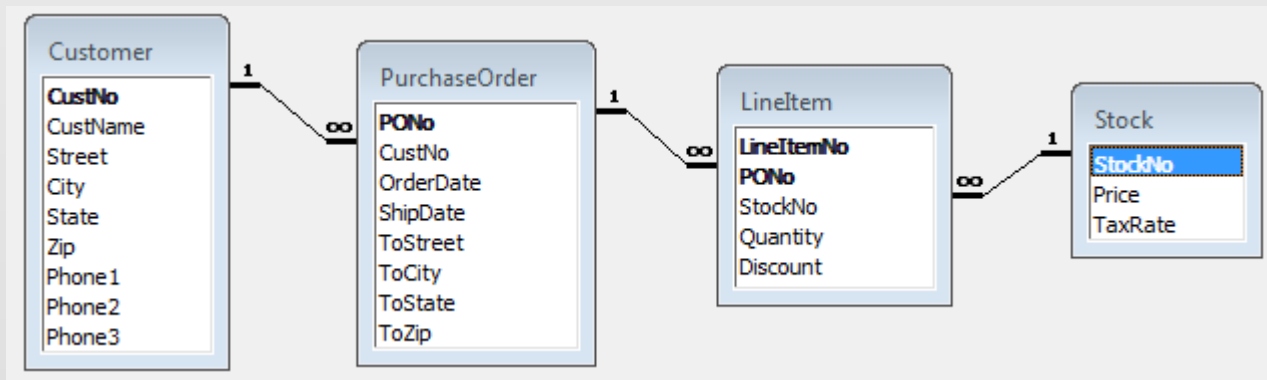
```
CREATE TABLE people_tab (  
    group_no NUMBER, people_column people_typ ) -- an instance of nested table  
NESTED TABLE people_column STORE AS people_column_nt;
```

Proiectarea unei baze de date în Oracle

- Se va compara proiectul relațional cu proiectul obiect-relațional al aceleiași baze de date – *Purchase Order (PO)*
- Baza de date este descrisă prin diagrama E/A de mai jos



- Schema logică relațională a bazei de date se obține ușor prin transpunerea directă a diagramei E/A:
 - tipurile de entitate din diagrama E/A devin relații (tabele) în schema logică
 - asocierile din diagrama E/A devin asocieri prin chei străine în schema logică



Crearea tabelelor în modelul relațional (1)

- Se creează tabelele și asocierile dintre ele:

```
CREATE TABLE Customer_reltab (  
    CustNo NUMBER NOT NULL,  
    CustName VARCHAR2(200) NOT NULL,  
    Street VARCHAR2(200) NOT NULL,  
    City VARCHAR2(200) NOT NULL,  
    State CHAR(2) NOT NULL,  
    Zip VARCHAR2(20) NOT NULL,  
    Phone1 VARCHAR2(20),  
    Phone2 VARCHAR2(20),  
    Phone3 VARCHAR2(20),  
    PRIMARY KEY (CustNo)  
);  
/  
  
CREATE TABLE Stock_reltab (  
    StockNo NUMBER PRIMARY KEY,  
    Price NUMBER,  
    TaxRate NUMBER  
);  
/
```

Crearea tabelelor în modelul relațional (2)

```
CREATE TABLE PurchaseOrder_reltab (  
    PONo NUMBER,                                /* PK, purchase order no */  
    Custno NUMBER references Customer_reltab,    /* FK referencing Customer */  
    OrderDate DATE,                             /* date of order */  
    ShipDate DATE,                              /* date to be shipped */  
    ToStreet VARCHAR2(200),                     /* shipto address */  
    ToCity VARCHAR2(200),  
    ToState CHAR(2),  
    ToZip VARCHAR2(20),  
    PRIMARY KEY(PONo)  
);  
/  
  
CREATE TABLE LinelItems_reltab (  
    LinelItemNo NUMBER,  
    PONo NUMBER REFERENCES PurchaseOrder_reltab, /* FK ref. PurchaseOrder*/  
    StockNo NUMBER REFERENCES Stock_reltab,      /* FK ref. Stock */  
    Quantity NUMBER,  
    Discount NUMBER,  
    PRIMARY KEY (PONo, LinelItemNo)  
);  
/
```

Inserarea datelor în tabelele relaționale

■ Stabilirea inventarului (stocurile):

```
INSERT INTO Stock_reltab VALUES(1004, 6750.00, 2);  
INSERT INTO Stock_reltab VALUES(1011, 4500.23, 2);  
INSERT INTO Stock_reltab VALUES(1534, 2234.00, 2);  
INSERT INTO Stock_reltab VALUES(1535, 3456.23, 2);
```

■ Înregistrarea clienților:

```
INSERT INTO Customer_reltab VALUES (1, 'Jean Nance', '2 Avocet Drive',  
    'Redwood Shores', 'CA', '95054', '415-555-0102', NULL, NULL);  
INSERT INTO Customer_reltab VALUES (2, 'John Nike', '323 College Drive',  
    'Edison', 'NJ', '08820', '609-555-0190', '201-555-0140', NULL);
```

■ Plasarea ordinelor de cumpărare:

```
INSERT INTO PurchaseOrder_reltab VALUES (1001, 1, SYSDATE, '10-MAY-2014',  
    NULL, NULL, NULL, NULL);  
INSERT INTO PurchaseOrder_reltab VALUES (2001, 2, SYSDATE, '20-MAY-2014',  
    '55 Madison Ave', 'Madison', 'WI', '53715');
```

■ Stabilirea liniilor din ordinele de cumpărare:

```
INSERT INTO LineItems_reltab VALUES(01, 1001, 1534, 12, 0);  
INSERT INTO LineItems_reltab VALUES(02, 1001, 1535, 10, 10);  
INSERT INTO LineItems_reltab VALUES(01, 2001, 1004, 1, 0);  
INSERT INTO LineItems_reltab VALUES(02, 2001, 1011, 2, 1);
```


Continutul tabelelor relaționale

Start Page x | Alex_PO x | CUSTOMER_RELTAB x

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	CUSTNO	CUSTNAME	STREET	CITY	STATE	ZIP	PHONE1	PHONE2	PHONE3
1	1	Jean Nance	2 Avocet Drive	Redwood Shores	CA	95054	415-555-0102	(null)	(null)
2	2	John Nike	323 College Drive	Edison	NJ	08820	609-555-0190	201-555-0140	(null)

Start Page x | Alex_PO x | STOCK_RELTAB x

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	STOCKNO	PRICE	TAXRATE
1	1004	6750	2
2	1011	4500.23	2
3	1534	2234	2
4	1535	3456.23	2

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	PONO	CUSTNO	ORDERDATE	SHIPDATE	TOSTREET	TOCITY	TOSTATE	TOZIP
1	1001	1	26-MAR-14	10-MAY-14	(null)	(null)	(null)	(null)
2	2001	2	26-MAR-14	20-MAY-14	55 Madison Ave	Madison	WI	53715

Start Page x | Alex_PO x | LINEITEMS_RELTAB x

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL


Sort.. | Filter:















	LINEITEMNO	PONO	STOCKNO	QUANTITY	DISCOUNT
1	1	1	1001	12	0
2	2	2	1001	10	10
3	3	1	2001	1	0
4	4	2	2001	1011	2

Interogări în modelul relațional (1)

- Interogarea: Care sunt datele clientului și ale liniilor unui ordin de cumpărare dat (PONo = 1001)?

```
SELECT C.CustNo, C.CustName, C.Street, C.City, C.State,  
       C.Zip, C.phone1, C.phone2, C.phone3,  
       P.PONo, P.OrderDate,  
       L.StockNo, L.LineItemNo, L.Quantity, L.Discount  
FROM   Customer_reltab C,  
       PurchaseOrder_reltab P,  
       LineItems_reltab L  
WHERE  C.CustNo = P.CustNo  
       AND P.PONo = L.PONo  
       AND P.PONo = 1001;
```

 All Rows Fetched: 2 in 0.007 seconds

	 CUSTNO	 CUSTNAME	 STREET	 CITY	 STATE	 ZIP	 PHONE1	 PHONE2	 PH...	 PONO	 ORDERDATE	 STOCKNO	 LINEITEMNO	
1	1	Jean Nance	2 Avocet Drive	Redwood Sh...	CA	95054	415-555-0102	(null)	(null)	1001	02-MAR-10	1534	1	
2	1	Jean Nance	2 Avocet Drive	Redwood Sh...	CA	95054	415-555-0102	(null)	(null)	1001	02-MAR-10	1535	2	

Interogări în modelul relațional (2)

- Interogarea: Care este valoarea totală a fiecarui ordine de cumpărare?

```
SELECT P.PONo, SUM(S.Price * L.Quantity)
FROM   PurchaseOrder_reltab P, LineItems_reltab L, Stock_reltab S
WHERE  P.PONo = L.PONo AND L.StockNo = S.StockNo GROUP BY P.PONo;
```

The screenshot shows a database query tool interface. On the left, a tree view lists several tables: EMPLOYEE_TAB, LINEITEMS_RELTAB, PEOPLE_COLUMN_NT, PEOPLE_TAB, PERSONS, PERSONS_TAB, and PURCHASEORDER_RELTAB. The PURCHASEORDER_RELTAB table is expanded, showing its columns: PONO, CUSTNO, ORDERDATE, SHIPDATE, TOSTREET, TOCITY, TOSTATE, and TOZIP. On the right, a SQL query is displayed in a text area:

```
SELECT P.PONo, SUM(S.Price * L.Quantity)
FROM PurchaseOrder_reltab P,
LineItems_reltab L,
Stock_reltab S
WHERE P.PONo = L.PONo
AND L.StockNo = S.StockNo
GROUP BY P.PONo;
```

Below the query, the 'Query Result' tab is active, showing the results of the query. The results are displayed in a table with 2 columns: PONO and SUM(S.PRICE*L.QUANTITY). The table contains 2 rows of data:

PONO	SUM(S.PRICE*L.QUANTITY)
2001	15750.46
1001	61370.3

Interogări în modelul relațional (3)

- Interogarea: Care sunt datele ordinului de cumparare si a liniilor acestora care se refera la stocul cu nr. 1004?

```
SELECT P.PONo, P.CustNo, L.StockNo, L.LineItemNo, L.Quantity, L.Discount
FROM PurchaseOrder_reltab P, LineItems_reltab L
WHERE P.PONo = L.PONo AND L.StockNo = 1004;
```

The screenshot shows a database query tool interface. The top pane displays the following SQL query:

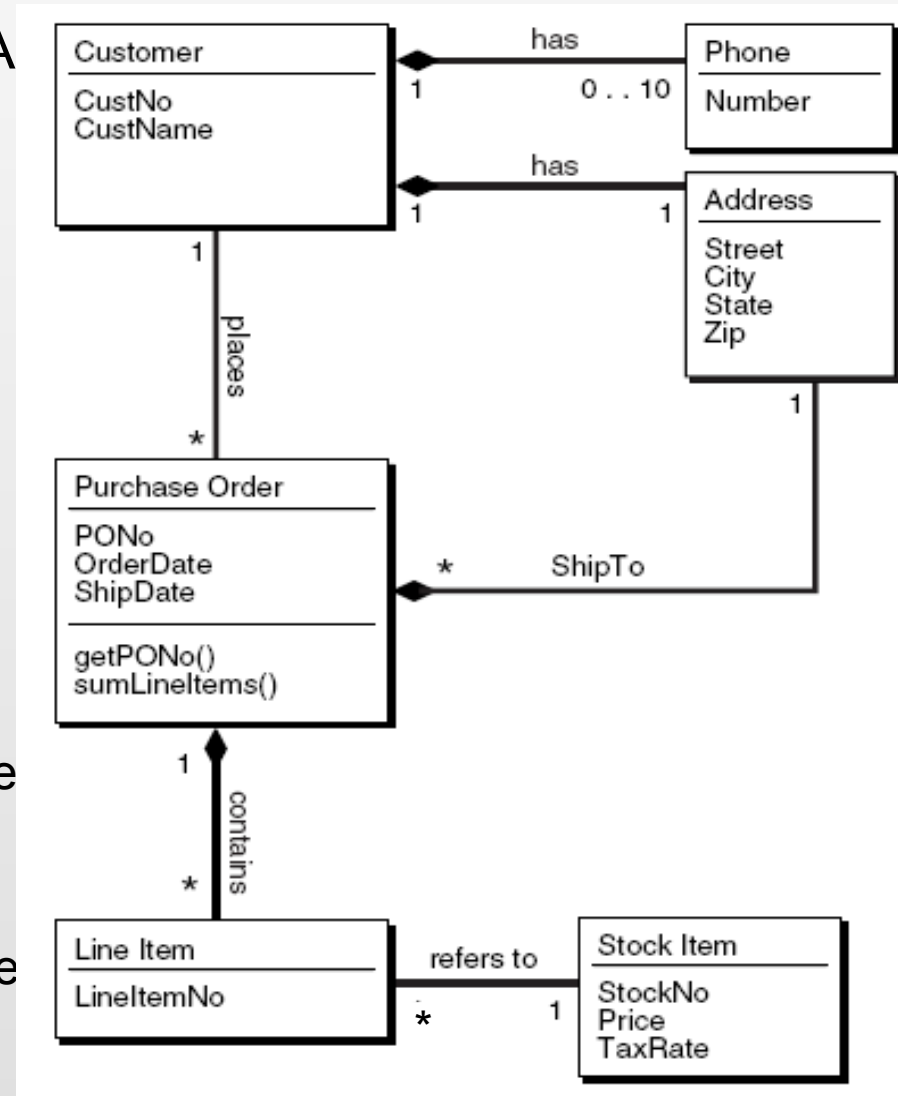
```
SELECT P.PONo, P.CustNo,
L.StockNo, L.LineItemNo, L.Quantity, L.Discount
FROM PurchaseOrder_reltab P,
LineItems_reltab L
WHERE P.PONo = L.PONo
AND L.StockNo = 1004;
```

Below the query editor, there are tabs for 'Script Output', 'Statement Output', and 'Query Result'. The 'Query Result' tab is active, showing the results of the query. The status bar indicates 'All Rows Fetched: 1 in 0.005 seconds'.

	PONO	CUSTNO	STOCKNO	LINEITEMNO	QUANTITY	DISCOUNT
1	2001	2	1004	1	1	0

Proiectarea schemei obiect-relaționale

- Plecând de la aceeași diagramă E/A se reprezintă schema obiect-relațională ca diagramă a claselor în limbajul UML:
- Tipurile de entități principale devin tipuri de date (“object type”): Customer, PurchaseOrder, Stock
- Se definește un **tip UDT** pentru adresă, care reunește toate componentele adresei (street, city etc.);
- Pentru reprezentarea telefoanelor se folosește un vector (***varray***)
- Pentru reprezentarea liniilor (LineItems) din ordinul de cumpărare (PurchaseOrder) se folosește un tabel imbricat (***nested table***)



Definirea tipurilor de date (1)

- Tipul *StockItem_objtyp* se folosește pentru definirea tabelului de obiecte “stock”
CREATE TYPE StockItem_objtyp AS OBJECT (
 StockNo NUMBER,
 Price NUMBER,
 TaxRate NUMBER);
- Tipul *LineItem_objtyp* se folosește pentru definirea unei linii de ordin de cump.
CREATE TYPE LineItem_objtyp AS OBJECT (
 LineItemNo NUMBER,
 Stock_ref REF StockItem_objtyp,
 Quantity NUMBER,
 Discount NUMBER);
- Tipul pentru tabelul imbricat cu liniile din ordinul de cumpărare (line items):
CREATE TYPE LineItemList_ntabtyp AS TABLE OF LineItem_objtyp;
- Tipul *Address_objtyp* se folosește pentru adrese:
CREATE TYPE Address_objtyp AS OBJECT (
 Street VARCHAR2(200),
 City VARCHAR2(200),
 State CHAR(2),
 Zip VARCHAR2(20));

Definirea tipurilor de date (2)

- Pentru numerele de telefon se definește un tip de date vector (varray):
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20);
- Tipul *Customer_objtyp* se folosește pentru definirea tabelului Customer
 - conține un obiect de tipul PhoneList_vartyp care este un vector pentru telefoane și un obiect de tipul *Address_objtyp* pentru adresa
 - conține o metodă de comparație de tip ORDER, care este invocată automat atunci când se compară două obiecte de acest tip

```
CREATE TYPE Customer_objtyp AS OBJECT (  
    CustNo NUMBER,  
    CustName VARCHAR2(200),  
    Address_obj Address_objtyp,  
    PhoneList_var PhoneList_vartyp,  
    ORDER MEMBER FUNCTION  
        compareCustOrders(x IN Customer_objtyp) RETURN INTEGER  
    ) NOT FINAL;  
/  
CREATE OR REPLACE TYPE BODY Customer_objtyp AS  
    ORDER MEMBER FUNCTION  
        compareCustOrders (x IN Customer_objtyp) RETURN INTEGER IS  
        BEGIN RETURN CustNo - x.CustNo; END;  
END;  
/
```

Definirea tipurilor de date (3)

- Crearea tipului PurchaseOrder_objtyp pentru tabelul PurchaseOrder:

```
CREATE TYPE PurchaseOrder_objtyp AUTHID CURRENT_USER AS OBJECT (  
    PONo NUMBER,  
    Cust_ref REF Customer_objtyp,  
    OrderDate DATE,  
    ShipDate DATE,  
    LineltemList_ntab LineltemList_ntabtyp,  
    ShipToAddr_obj Address_objtyp,  
    MAP MEMBER FUNCTION getPONo RETURN NUMBER,  
    MEMBER FUNCTION sumLineltems RETURN NUMBER);  
/
```

- Fiecare obiect instanță a acestui tip reprezintă un ordin de cumpărare (PurchaseOrder). Obiectele au șase atribute care sunt:
 - atribute de tipuri predefinite (PONo, OrderDate, ShipDate)
 - o referință la un obiect de tipul Customer_objtyp, care asigură asocierea cu tabelul Customer
 - un obiect de tipul Adress_objtyp, care conține adresa de livrare (ShipTo)
 - un tabel imbricat de tipul LineltemList_ntabtyp, care conține liniile ordinului

Definirea tipurilor de date (4)

- Metodele tipului PurchaseOrder_objtyp, definite ca funcții PL/SQL sunt:
 - O metodă de comparație de tipul MAP (getPONo) care returnează valoarea atributului PONo și care va fi invocată automat ori de câte ori se compară două obiecte de acest tip
 - O metodă de calcul a sumei valorilor liniilor unui ordin (sumLineItems)

```
CREATE OR REPLACE TYPE BODY PurchaseOrder_objtyp AS
  MAP MEMBER FUNCTION getPONo RETURN NUMBER IS
  BEGIN RETURN PONo; END;
  MEMBER FUNCTION sumLineItems RETURN NUMBER IS
  i INTEGER;
  StockVal StockItem_objtyp;
  Total NUMBER := 0;
BEGIN
  FOR i in 1..SELF.LineItemList_ntab.COUNT LOOP
    UTL_REF.SELECT_OBJECT(LineItemList_ntab(i).Stock_ref,StockVal);
    Total := Total + SELF.LineItemList_ntab(i).Quantity * StockVal.Price;
  END LOOP;
RETURN Total;
END;
END;
/
```

Definirea tipurilor de date (5)

- La definirea metodei `sumLineItems` apar următoarele elemente:
 - Cuvântul cheie `SELF` indică utilizarea atributelor proprii obiectului
 - Cuvântul cheie `COUNT` specifică numărul de elemente ale tabelului
 - Pentru dereferențierea referinței `Stock_ref` se apelează funcția `SELECT_OBJECT` din package-ul `UTL_REF`, deoarece PL/SQL nu suportă dereferențierea directă din programe PL/SQL
- Sintaxa `AUTHID CURRENT_USER` specifică faptul ca tipul `PurchaseOrder_objtyp` este definit folosind drepturile utilizatorului curent

Crearea tabelelor de obiecte

- Folosind tipurile UDT definite anterior se vor crea urm. tabele de obiecte:
 - Stock_objtab:
CREATE TABLE Stock_objtab OF StockItem_objtyp (StockNo PRIMARY KEY)
OBJECT IDENTIFIER IS PRIMARY KEY;
 - Customer_objtab
 - PurchaseOrder_objtab
 - LineItemList_ntab (tabel imbricat în PurchaseOrder_objtab)
- Fiecare linie a unui astfel de tabel conține un obiect de tipul specificat (obiect linie)
- Obiectele linie sunt referențiabile, adică alte linii relaționale sau obiecte linii (row objects) pot referi un obiect linie folosind identif. acesteia (OID)
- Identificatorii OID ai obiectelor linie:
 - pot fi generați automat de sistem; aceasta se specifică în instrucțiunea **CREATE TABLE** prin opțiunea: **OBJECT IDENTIFIER IS SYSTEM GENERATED**
 - se poate folosi cheia primară a tabelului ca identificator OID al obiectelor tabelului; aceasta se specifică în instrucțiunea **CREATE TABLE** prin opțiunea: **OBJECT IDENTIFIER IS PRIMARY KEY**

Tabelul Customer_objtab

**CREATE TABLE Customer_objtab OF Customer_objtyp (CustNo PRIMARY KEY)
OBJECT IDENTIFIER IS PRIMARY KEY;**

Fiecare obiect linie
conține:

- Atribute de tipuri predefinite (CUSTNO, CUSTNAME)
- Un atribut obiect de tipul *Addres_objtyp* care conține componentele adresei
- Un atribut obiect de tipul *PhoneList_vartyp*, care este un vector de dimensiune maximă 10 cu numere de telefon

Object Relational Representation of Table Customer_objtab

Table CUSTOMER_OBJTAB (of CUSTOMER_OBJTYP)			
CUSTNO	CUSTNAME	ADDRESS_OBJ	PHONELIST_VAR
Number NUMBER	Text VARCHAR2(200)	Object Type ADDRESS_OBJTYP	Varray PHONELIST_VARTYP
PK			

Varray PHONELIST_VAR (of PHONELIST_VARTYP)	
(PHONE)	
Number NUMBER	

Column Object ADDRESS_OBJ (of ADDRESS_OBJTYP)			
STREET	CITY	STATE	ZIP
Text VARCHAR2(200)	Text VARCHAR2(200)	Text CHAR(2)	Number VARCHAR2(20)
PK			

Tabelul PurchaseOrder_objtab (1)

```
CREATE TABLE PurchaseOrder_objtab OF PurchaseOrder_objtyp ( /* Line 1 */
    PRIMARY KEY (PONo), /* Line 2 */
    FOREIGN KEY (Cust_ref) REFERENCES Customer_objtab) /* Line 3 */
    OBJECT IDENTIFIER IS PRIMARY KEY /* Line 4 */
    NESTED TABLE LineltemList_ntab STORE AS PoLine_ntab ( /* Line 5 */
    (PRIMARY KEY(NESTED_TABLE_ID, LineltemNo)) /* Line 6 */
    ORGANIZATION INDEX COMPRESS) /* Line 7 */
    RETURN AS LOCATOR /* Line 8 */
/
```

- **Line 1:** specifică tipul obiectelor tabelului; fiecare linie va conține un obiect de tipul PurchaseOrder_objtyp; attributele acestor obiecte sunt:

PONo	NUMBER
Cust_ref	REF Customer_objtyp
OrderDate	DATE
ShipDate	DATE
LineltemList_ntab	LineltemList_ntabtyp
ShipToAddr_obj	Address_objtyp

- **Line 2:** specifică cheia primară a tabelului (PONo)

Tabelul PurchaseOrder_objtab (2)

- **Line 3: FOREIGN KEY (Cust_ref) REFERENCES Customer_objtab)**
 - Specifică constrângerea de integritate referențială (FOREIGN KEY): atributul Cust_ref trebuie să refere numai obiecte existente în tabelul Customer_objtab

Object Relational Representation of Table PurchaseOrder_objtab

Table PURCHASEORDER_OBJTAB (of PURCHASEORDER_OBJTYP)					
PONO	CUST_REF	ORDERDATE	SHIPDATE	LINEITEMLIST_NTAB	SHIPTOADDR_OBJ
Number NUMBER	Reference CUSTOMER_ OBJTYP	Date DATE	Date DATE	Nested Table LINEITEMLIST_ NTABTYP	Object Type ADDRESS_ OBJTYP
PK	FK				
<div> MEMBER FUNCTION getPONO RETURN NUMBER MEMBER FUNCTION SumLineItems RETURN NUMBER </div>					

Reference
to a row of
the table

Column object of
the defined type

Table CUSTOMER_OBJTAB (of CUSTOMER_OBJTYP)			
CUSTNO	CUSTNAME	ADDRESS_OBJ	PHONELIST_VAR
Number NUMBER	Text VARCHAR2(200)	Object Type ADDRESS_OBJTYP	Varray PHONELIST_VARTYP
PK			

Column Object SHIPTOADDR_OBJ (of ADDR_OBJTYP)			
STREET	CITY	STATE	ZIP
Text VARCHAR2(200)	Text VARCHAR2(200)	Text CHAR(2)	Number VARCHAR2(20)

Tabelul PurchaseOrder_objtab (3)

■ Line 4: OBJECT IDENTIFIER IS PRIMARY KEY

- Această linie specifică faptul că identificatorul OID al unui obiect linie al tabelului este cheia primară a tabelului

■ Line 5-8: specifică proprietățile tabelului imbricat LineltemList_ntab

```
NESTED TABLE LineltemList_ntab STORE AS PoLine_ntab (      /* Line 5 */  
    (PRIMARY KEY(NESTED_TABLE_ID, LineltemNo))              /* Line 6 */  
    ORGANIZATION INDEX COMPRESS)                             /* Line 7 */  
    RETURN AS LOCATOR                                         /* Line 8 */
```

- Line 5: indică faptul că tabelul imbricat LineltemList_ntab este memorat ca tabel separat, numit PoLine_ntab

- Linia 6: o coloană ascunsă numită NESTED_TABLE_ID pune în corespondență fiecare linie din tabelul imbricat cu linia părinte

- Linia 7: tabelul imbricat este organizat indexat (index-organized table - IOT) cu compresia identificatorilor (adică identificatorul liniei părinte se memorează o singură dată pentru toate liniile imbricate corespunzătoare)

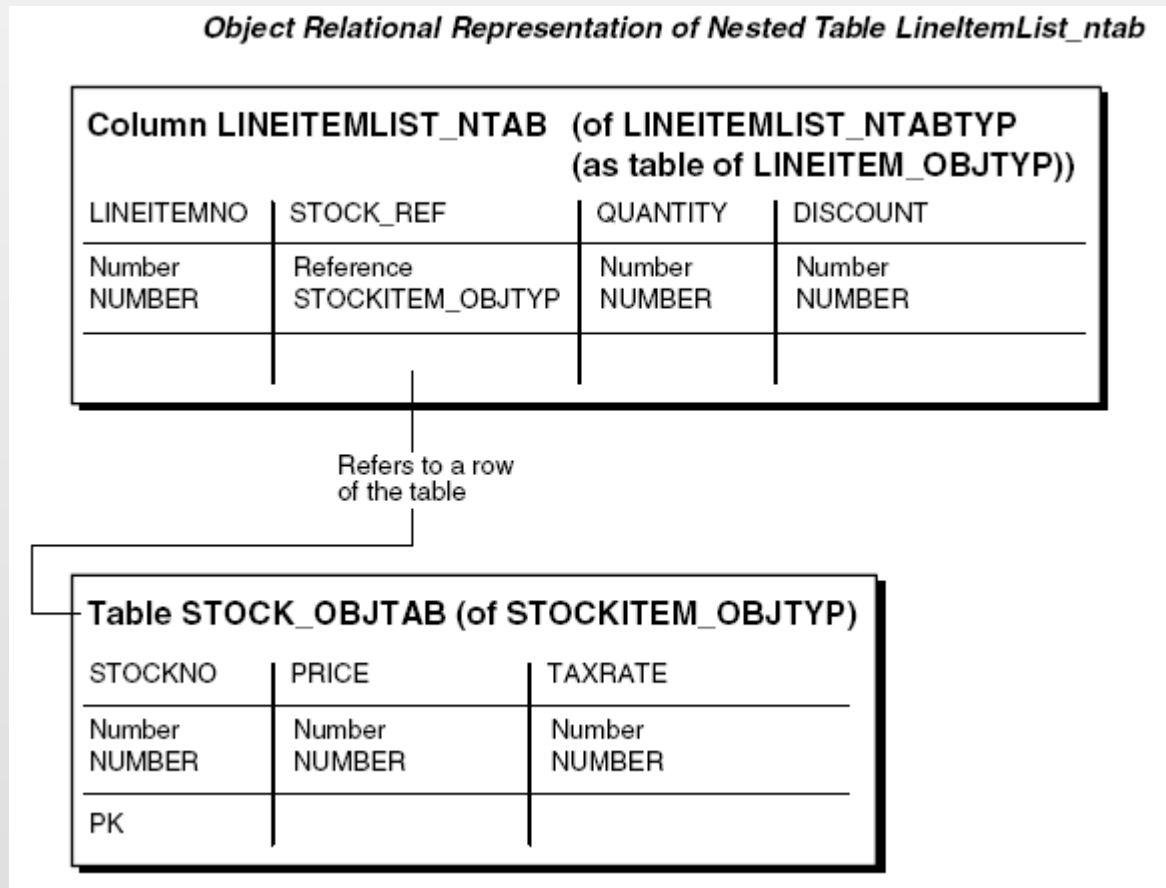
- Linia 8: se returnează locația liniei imbricate; dacă acest specificator lipsește, implicit se returnează VALUE, adică tot tabelul imbricat

■ Se adaugă SCOPE pentru STOCK_REF:

```
ALTER TABLE PoLine_ntab  
    ADD (SCOPE FOR (Stock_ref) IS Stock_objtab);
```

Tabelul imbricat LineltemList_ntab

- Tabelul imbricat LineltemList_ntab conține o referință (STOCK_REF) la tabelul Stock_objtab, care realizează asocierea între tabelul PurchaseOrder_objtab și tabelul Stock_objtab



Inserarea datelor în tabelele Stock_objtab și Customer_objtab

- Inserarea datelor în tabelele de obiecte seamănă cu inserarea în tabelele relaționale, doar că pentru fiecare obiect se invocă constructorul
- Inserarea datelor în tabelul Stock_objtab:

```
INSERT INTO Stock_objtab VALUES(1004, 6750.00, 2) ;
INSERT INTO Stock_objtab VALUES(1011, 4500.23, 2) ;
INSERT INTO Stock_objtab VALUES(1534, 2234.00, 2) ;
INSERT INTO Stock_objtab VALUES(1535, 3456.23, 2) ;
```
- Inserarea datelor în tabelul Customer_objtab:

```
INSERT INTO Customer_objtab
VALUES (1, 'Jean Nance',
        Address_objtyp('2 Avocet Drive', 'Redwood Shores', 'CA', '95054'),
        PhoneList_vartyp('415-555-0102')
);
INSERT INTO Customer_objtab
VALUES ( 2, 'John Nike',
        Address_objtyp('323 College Drive', 'Edison', 'NJ', '08820'),
        PhoneList_vartyp('609-555-0190','201-555-0140')
);
```

Conținutul tabelelor Stock_objtab și Customer_objtab

Start Page x | Alex_PO x | STOCK_OBJTAB x

Columns | Data | Constraints | Grants | Statistics | Triggers

Sort: | Filter:

	STOCKNO	PRICE	TAXRATE
1	1004	6750	2
2	1011	4500.23	2
3	1534	2234	2
4	1535	3456.23	2

Tables (Filtered)

- CUSTOMER_OBJTAB
 - CUSTNO
 - CUSTNAME
 - ADDRESS_OBJ
 - PHONELIST_VAR
- CUSTOMER_RELTAB
- LINEITEMS_RELTAB
- POLINE_NTAB
- PURCHASEORDER_OBJTAB
- PURCHASEORDER_RELTAB
- STOCK_OBJTAB
 - STOCKNO
 - PRICE
 - TAXRATE
- STOCK_RELTAB

Views

Editing Views

Indexes

Packages

	CUSTNO	CUSTNAME	ADDRESS_OBJ	PHONELIST_VAR
1	1	Jean Nance	[PO.ADDRESS_OBJTYP]	PO.PHONELIST_VARTYP('415-555-0102')
2	2	John Nike	[PO.ADDRESS_OBJTYP]	PO.PHONELIST_VARTYP('609-555-0190', '201-555-0140')

Single Record View

Navigation: << < > >>

CUSTNO: 1

CUSTNAME: Jean Nance

ADDRESS_OBJ: PO.ADDRESS_OBJTYP('2 Avocet Drive', 'Redwood Shores', 'CA', '95054')

PHONELIST_VAR: PO.PHONELIST_VARTYP('415-555-0102')

Buttons: Help, Cancel

Inserarea datelor în tabelul PurchaseOrder_objtab

- Inserarea datelor în tabelul PurchaseOrder_objtab:

```
INSERT INTO PurchaseOrder_objtab
  SELECT 1001, REF(C), SYSDATE, '10-MAY-14',
  LineltemList_ntabtyp(), NULL
  FROM Customer_objtab C WHERE C.CustNo = 1 ;
INSERT INTO PurchaseOrder_objtab
  SELECT 2001, REF(C), SYSDATE, '20-MAY-14', LineltemList_ntabtyp(),
  Address_objtyp('55 Madison Ave','Madison','WI','53715')
  FROM Customer_objtab C WHERE C.CustNo = 2 ;
```

- Prima instrucțiune crează un obiect de tipul PurchaseOrder_objtyp cu următoarele atribute:

PONo	1001
Cust_ref referință la Customer cu CustNo = 1	
DrderDate	SYSDATE
ShipDate	10-May-14
LineltemList_ntab	o listă vidă de obiecte de tipul Lineltem_ntabtyp
ShipToAddr_obj	null

- Aceaste instrucțiuni folosesc o interogare SELECT pentru a construi referința la obiectul din tabelul Customer_objtab care are CustNo = ...

Inserarea datelor în tabelul LineltemList_ntab (1)

- Următoarele instrucțiuni de inserare identifică tabelul imbricat ca țintă pentru inserare, și anume tabelul imbricat în coloana LineltemList_ntab a obiectului din tabelul PurchaseOrder_objtab si linia care are valoarea lui PONo egală cu o valoare data:

```
INSERT INTO TABLE (  
    SELECT P.LineltemList_ntab  
    FROM PurchaseOrder_objtab P  
    WHERE P.PONo = 1001  
)  
SELECT 01, REF(S), 12, 0 FROM Stock_objtab S  
WHERE S.StockNo = 1534 ;
```

```
INSERT INTO TABLE (  
    SELECT P.LineltemList_ntab  
    FROM PurchaseOrder_objtab P  
    WHERE P.PONo = 1001  
)  
SELECT 02, REF(S), 10, 10  
FROM Stock_objtab S  
WHERE S.StockNo = 1535 ;
```

Inserarea datelor în tabelul LineltemList_ntab (2)

```
INSERT INTO TABLE (  
    SELECT P.LineltemList_ntab  
    FROM PurchaseOrder_objtab P  
    WHERE P.PONo = 2001  
)  
SELECT 10, REF(S), 1, 0  
FROM Stock_objtab S  
WHERE S.StockNo = 1004 ;
```

```
INSERT INTO TABLE (  
    SELECT P.LineltemList_ntab  
    FROM PurchaseOrder_objtab P  
    WHERE P.PONo = 2001  
)  
VALUES(11, (SELECT REF(S)  
FROM Stock_objtab S  
WHERE S.StockNo = 1011), 2, 1  
);
```

- Valoarea inserată conține o referință la obiectul linie din tabelul STOCK_objtab care are valoarea atributului StockNo dorită

Conținutul tabelului PurchaseOrder_objtab

	PONO	CUST_REF	ORDERDATE	SHIPDATE	LINEITEMLIST_NTAB	SHIPTOADDR_OBJ
1	1001	[PO.CUSTOMER_OBJTYP]	26-MAR-14	10-MAY-14	PO.LINEITEMLIST_NTABTYP([PO.LIN...	(null)
2	2001	[PO.CUSTOMER_OBJTYP]	26-MAR-14	20-MAY-14	PO.LINEITEMLIST_NTABTYP([PO.LIN...	[PO.ADDRESS_OBJTYP]

Single Record View



PONO	<input type="text" value="1001"/>	
CUST_REF	<input type="text" value=".ADDRESS_OBJTYP('2 Avocet Drive','Redwood Shores','CA','95054'),PO.PHONELIST_VARTYP(415-555-0102))"/>	
ORDERDATE	<input type="text" value="26-MAR-14"/>	
SHIPDATE	<input type="text" value="10-MAY-14"/>	
LINEITEMLIST_NTAB	<input type="text" value="_OBJTYP(1,'oracle.sql.REF@d870bbfc',12,0),PO.LINEITEM_OBJTYP(2,'oracle.sql.REF@d870bbfc',10,10)"/>	
SHIPTOADDR_OBJ	<input type="text"/>	

Interogări în baza de date obiect-relațională

- Interogarea: Care sunt datele ordinelor de cumpărare, ordonate după numărul ordinului?

```
SELECT p.PONo FROM PurchaseOrder_objtab p ORDER BY VALUE(p) ;
```

	PONO
1	1001
2	2001

- În această interogare este invocată automat metoda getPONo pentru ordonarea liniilor rezultatului după atributul PONo

- Interogarea: Care este valoarea totală a fiecărui ordin de cumpărare?

```
SELECT p.PONo, p.sumLineItems()  
FROM PurchaseOrder_objtab p ;
```

	PONO	P.SUMLINEITEMS()
1	1001	61370.3
2	2001	15750.46

- Interogarea: Care sunt datele clientului și detaliile (liniile) ordinului de cumpărare cu numărul 1001?

```
SELECT Deref(p.Cust_ref), p.ShipToAddr_obj, p.PONo, p.OrderDate,  
LineItemList_ntab FROM PurchaseOrder_objtab p WHERE p.PONo = 1001 ;
```

DEREF(P.CUST_REF)	SHIPTOADDR_OBJ	PONO	ORDERDATE	LINEITEMLIST_NTAB
[PO.CUSTOMER_OBJTYP]	{null}	1001	26-MAR-14	PO.LINEITEMLIST_NTABTYP([PO.LINEITEM_OBJTYP],[PO.LINEITEM_OBJTYP])

Single Record View

DEREF(P.CUST_REF) :an Nance',PO.ADDRESS_OBJTYP('2 Avocet Drive','Redwood Shores','CA','95054'),PO.PHONELIST_VARTYP(415-555-0102))

SHIPTOADDR_OBJ

PONO 1001

ORDERDATE 26-MAR-14

Comparație – baze de date relaționale și obiect-relaționale Oracle

- Exemplul de mai sus a avut ca scop comparația între proiectul relațional și cel obiect-relațional al aceleiași baze de date *Purchase Order* (PO)
- Proiectul relațional al unei baze de date pare mai simplu decât proiectul obiect-relațional al aceleiași baze de date, dar modelul obiect-relațional:
 - Structurează proiectul bazei de date – prin gruparea atributelor în tipuri și vectori de date (tip adresa, vector de numere de telefon etc.)
 - Folosește referințe pentru asociere, în locul cheilor străine, ceea ce este mult mai eficient
 - Permite gruparea unor linii de asociere în tabele imbricate, mărin­d viteza de execuție (nu se mai caută valoarea referită în tabel ci se folosește indexarea)
 - În modelul relațional asocierea între tabelul *PurchaseOrder_reltab* și *Stock_reltab* se face prin tabel *LineItems_reltab* care conține 2 chei străine
 - În modelul obiect-relațional, asocierea dintre tabelul *PurchaseOrder_objtab* și *Stock_objtab* se dă printr-un tabel imbricat *LineItemList_ntab* (conținut ca atribut al tipului *PurchaseOrder_objtyp*), compus din liniile ordinului de plată respectiv, și fiecare linie din tabelul imbricat conține o referință la un obiect din tabelul *Stock_objtab*
- Majoritatea bazelor de date pentru aplicații științifice folosesc caracteristicile modelului obiect-relațional, atât pentru tipuri complexe de date (tipuri de date spațiale, temporale, multimedia etc.) cât și pentru referințe, vectori, tabele imbricate

Caracteristici obiect-relaționale în alte SGBD-uri

- MySQL (versiunea 5.6) nu oferă suport pentru tipuri definite de utilizator, dar are unele extensii de tipuri de date mai complexe, pentru format XML și date spațiale (vor fi prezentate la capitolele corespunzătoare)
- PostgreSQL (versiunea 8.4) are o varietate de tipuri complexe predefinite sau definite de utilizator:
 - Geometric types
 - Network address types
 - Arrays types
 - Composite types
 - Object identifier types – folosite de PostgreSQL ca și chei primare în tabelele de sistem
 - Pseudo types – tipuri care pot fi returnate de funcții (nu coloane în tabele)
 - Tipuri definite de utilizator

Tipuri geometrice în PostgreSQL

- Tipurile geometrice sunt predefinite, impreuna cu mai multi operatori geometrici

Geometric Types

Name	Storage Size	Representation	Description
point	16 bytes	Point on a plane	(x,y)
line	32 bytes	Infinite line (not fully implemented)	((x1,y1),(x2,y2))
lseg	32 bytes	Finite line segment	((x1,y1),(x2,y2))
box	32 bytes	Rectangular box	((x1,y1),(x2,y2))
path	16+16n bytes	Closed path (similar to polygon)	((x1,y1),...)
path	16+16n bytes	Open path	[(x1,y1),...]
polygon	40+16n bytes	Polygon (similar to closed path)	((x1,y1),...)
circle	24 bytes	Circle	<(x,y),r> (center point and radius)

Operator	Description	Example
+	Translation	box '((0,0),(1,1))' + point '(2.0,0)'
-	Translation	box '((0,0),(1,1))' - point '(2.0,0)'
*	Scaling/rotation	box '((0,0),(1,1))' * point '(2.0,0)'
/	Scaling/rotation	box '((0,0),(2,2))' / point '(2.0,0)'
#	Point or box of intersection	'((1,-1),(-1,1))' # '((1,1),(-1,-1))'

Tipuri complexe în PostgreSQL

- Tablouri uni și multidimensionale de date și tabele care contin tablouri:

```
CREATE TABLE tictactoe (  
    squares integer[ ][ ]);
```
- Tipuri compuse și tabele care conțin tipuri compuse:

```
CREATE TYPE complex AS (  
    r double precision,  
    i double precision );  
CREATE TYPE inventory_item AS (  
    name text, -- text este un sir de caractere de lungime nelimitata, delimitat caract. spec.  
    supplier_id integer,  
    price numeric );  
CREATE TABLE inventory_tab (  
    item inventory_item,  
    count integer );
```
- Se pot executa operații de manipulare a datelor folosind instrucțiuni SQL

```
INSERT INTO inventory_tab VALUES (ROW('fuzzy dice', 42, 1.99), 1000);  
SELECT item.name FROM inventory_tab WHERE item.price > 9.99;
```

Tipuri definite de utilizator în PostgreSQL (1)

- Tipurile definite de utilizator:
 - Conțin tipuri de date (structuri, clase) și metode create într-un limbaj de uz general (tipic, C), memorate într-o bibliotecă partajată (so, dll)
 - Tipul SQL corespunzător accesează și utilizează biblioteca respectivă
- Pentru fiecare tip definit de utilizator se definesc funcții de intrare și ieșire care specifică modul de serializare a tipului, precum și funcții pentru orice operație efectuată cu obiecte de acel tip
 - Funcția de intrare primește ca argument un șir de caractere terminat cu null și returnează reprezentarea internă (în memorie) a tipului
 - Funcția de ieșire primește ca argument reprezentarea internă a unui obiect de acel tip și returnează un șir de caractere terminat cu null
- Exemplu: se definește în C tipul (structura) de numere complexe și funcțiile de intrare ieșire într-o bibliotecă partajată (în 'filename'):

```
typedef struct Complex {  
    double x;  
    double y;  
} Complex;
```

Tipuri definite de utilizator în PostgreSQL (2)

- Apoi se declară tipul complex și funcțiile de intrare-ieșire în SQL:

```
CREATE TYPE complex;
```

```
CREATE FUNCTION complex_in(cstring)  
  RETURNS complex  
  AS 'filename'  
  LANGUAGE C IMMUTABLE STRICT;
```

```
CREATE FUNCTION complex_out(complex)  
  RETURNS cstring  
  AS 'filename'  
  LANGUAGE C IMMUTABLE STRICT;
```

IMMUTABLE STRICT – înseamnă ca funcția nu poate modifica baza de date

- În final se poate defini tipul SQL complex:

```
CREATE TYPE complex (  
  internallength = 16,  
  input = complex_in,  
  output = complex_out,  
  alignment = double  
);
```

Caracteristici obiect-relaționale în MS SQL Server

- Începând cu versiunea SQL Server 2005, tipurile de date definite de utilizator, procedurile stocate, funcțiile și triggererele sunt integrate în executivul CLR (*Common Language Runtime*) din platforma .NET, ceea ce înseamnă că se pot folosi toate limbajele .NET
- Programele sursă în C++ (Managed C++), C#, J#, Visual Basic .NET se compilează în module în limbajul MSIL (MS Intermediate Language)
- Executivul CLR este o mașină virtuală care gestionează execuția programelor .NET (a modulelor MSIL)
- Toate limbajele .NET respectă specificația de tipuri comune numită *Common Language Specification* (CLS)
- CLR nu operează direct cu module ci cu ansambluri (assemblies)
- Un ansamblu (assembly) este o componenta software sub forma de bibliotecă DLL sau o aplicație executabilă .exe, compusă dintr-o grupare logică de module și de resurse și este cea mai mică unitate de reutilizare, instalare și securitate
- În funcție de opțiunile de compilare un *assembly* este compus dintr-unul sau mai multe fișiere

Tipuri definite de utilizator în SQL Server

- Pentru crearea tipurilor definite de utilizator se urmează pașii:
 - Se creează tipul de date ca o clasă sau structură într-un limbaj .NET
 - Se compilează clasa cu compilatorul limbajului respectiv și se obține ansamblul corespunzător
 - Se înregistrează ansamblul în SQL Server cu comanda:
`CREATE ANSAMBLY`
 - Se creează tipul SQL care referă ansamblul înregistrat
- La deployment-ul unui proiect SQL Server în Microsoft Visual Studio se înregistrează automat în baza de date ansamblul specificat de proiect
- De asemenea, la deployment CLR generează câte un tip definit de utilizator pentru baza de date corespunzător oricărei clase care are adnotarea (atributul) `SqlUserDefinedType`
- Pentru ca SQL Server să execute cod CLR trebuie configurat folosind opțiunea `CLR-enabled` în procedura `sp_configure`

Concluzii – Baze de date obiect-relaționale

- SGBD-urile actuale tind să devină obiect-relaționale prin introducerea gradată a obiectelor într-o reprezentare relațională
- Avantajele oferite de modelul obiect-relațional (structurare, extensibilitate, reutilizabilitate, eficiență) sunt foarte importante pentru aplicații științifice, industriale și comerciale complexe
- Dar integrarea conceptelor relaționale cu cele obiect-orientate este o provocare care necesită multe eforturi de cercetare, deoarece cele două direcții de dezvoltare sunt diametral opuse
- Ca urmare, dificultatea de proiectare și implementare a bazelor de date OR este mult mai mare decât a celor pur relaționale, dar rezultatele proiectelor sunt mai structurate, mai extensibile și mai eficiente
- Bazele de date avansate (XML, spațiale, multimedia, medicale etc.) **SUNT** baze de date obiect-relaționale

Bibliografie specifică

Baze de date obiect-relaționale

- Oracle Documentation
 - Oracle SQL Language Reference
 - Oracle PL/SQL Language Reference
 - Oracle Object-Relational Developer's Guide