The picture can't be displayed.

Chapter 22: Object-Based Databases

Database System Concepts, 6th Ed.

Chapter 22: Object-Based Databases

- Complex Data Types and Object Orientation
- Structured Data Types and Inheritance in SQL
- Table Inheritance
- Array and Multiset Types in SQL
- Object Identity and Reference Types in SQL
- Implementing O-R Features
- Persistent Programming Languages
- Comparison of Object-Oriented and Object-Relational Databases

Object-Relational Data Models

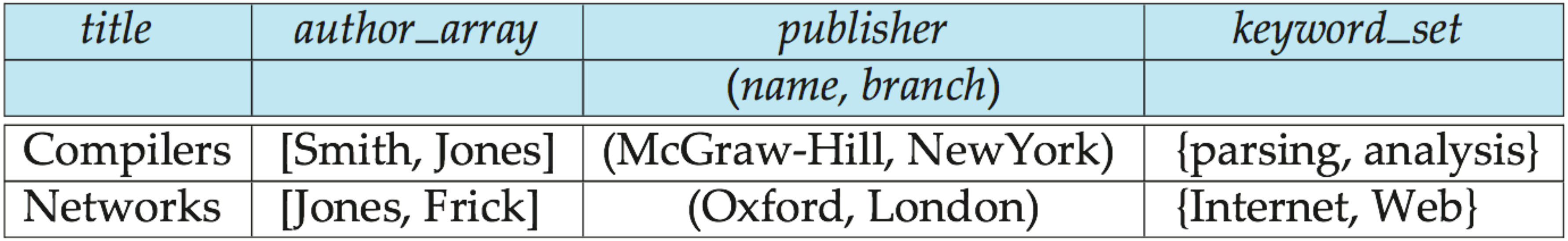
- Extend the relational data model by including object orientation and constructs to deal with added data types.
- Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
- Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
- Upward compatibility with existing relational languages.

Complex Data Types

- Motivation:
  - Permit non-atomic domains (atomic ≡ indivisible)
  - Example of non-atomic domain: set of integers,or set of tuples
  - Allows more intuitive modeling for applications with complex data
- Intuitive definition:
  - allow relations whenever we allow atomic (scalar) values — relations within relations
  - Retains mathematical foundation of relational model
  - Violates first normal form.

Example of a Nested Relation

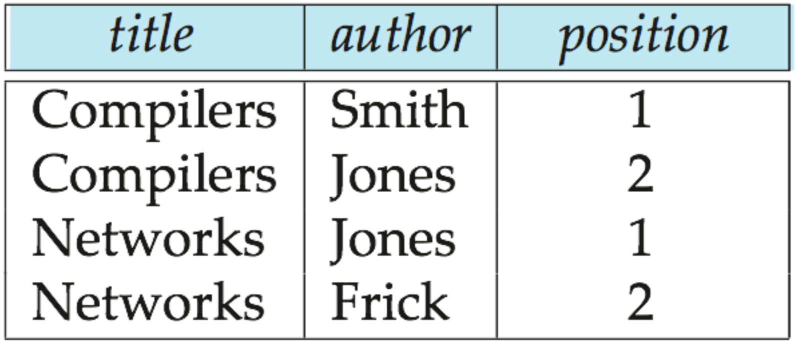
- Example: library information system
- Each book has
  - title,
  - a list (array) of authors,
  - Publisher, with subfields *name* and *branch*, and
  - a set of keywords
- Non-1NF relation *books*



title	*author_array*	*publisher*	*keyword_set*
		(name, branch)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

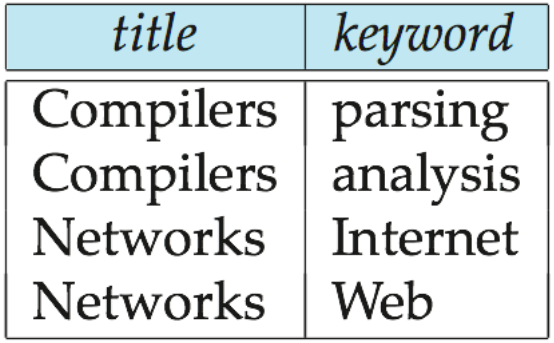
4NF Decomposition of Nested Relation

- Suppose for simplicity that title uniquely identifies a book
  - In real world ISBN is a unique identifier
- Decompose *books* into 4NF using the schemas:
  - (*title, author, position*)
  - (*title, keyword*)
  - (*title, pub-name, pub-branch*)
- 4NF design requires users to include joins in their queries.



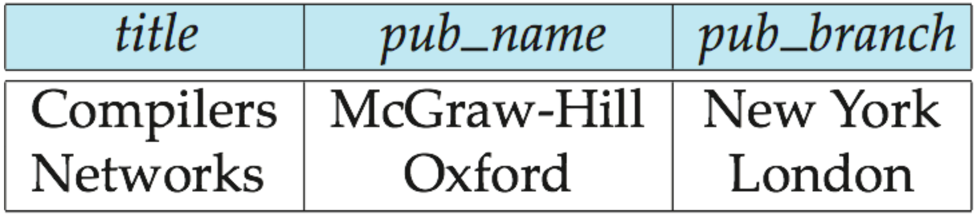
title	*author*	*position*
Compilers	Smith	1
Compilers	Jones	2
Networks	Jones	1
Networks	Frick	2

authors



title	*keyword*
Compilers	parsing
Compilers	analysis
Networks	Internet
Networks	Web

keywords



title	*pub_name*	*pub_branch*
Compilers	McGraw-Hill	New York
Networks	Oxford	London

books4

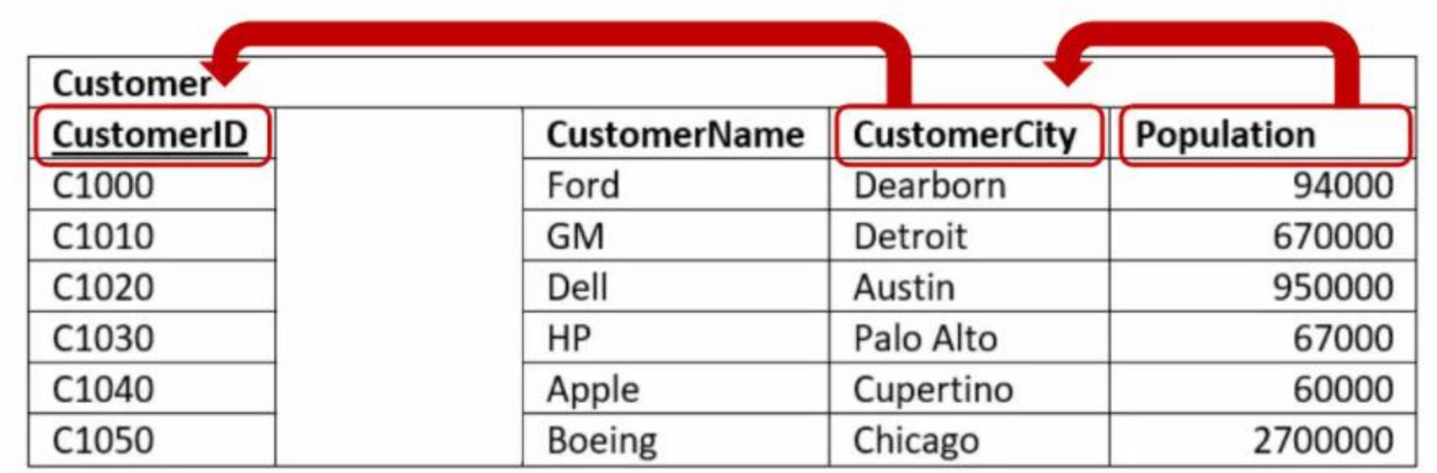
1NF, 2NF, 3NF, BCNF, 4NF

- 1NF (First Normal Form) Rules
  - Each table cell should contain a single value.
  - Each record needs to be unique.

- A table is in 2nd Normal Form if:
  - The table is in 1st normal form, and
  - All the non-key columns are dependent on the table’s primary key (Avoids repetitions of data in tables).

- A table is in third normal form if:
  - A table is in 2nd normal form.
  - It contains only columns that are non-transitively dependent on the primary key

1NF, 2NF, 3NF, BCNF, 4NF



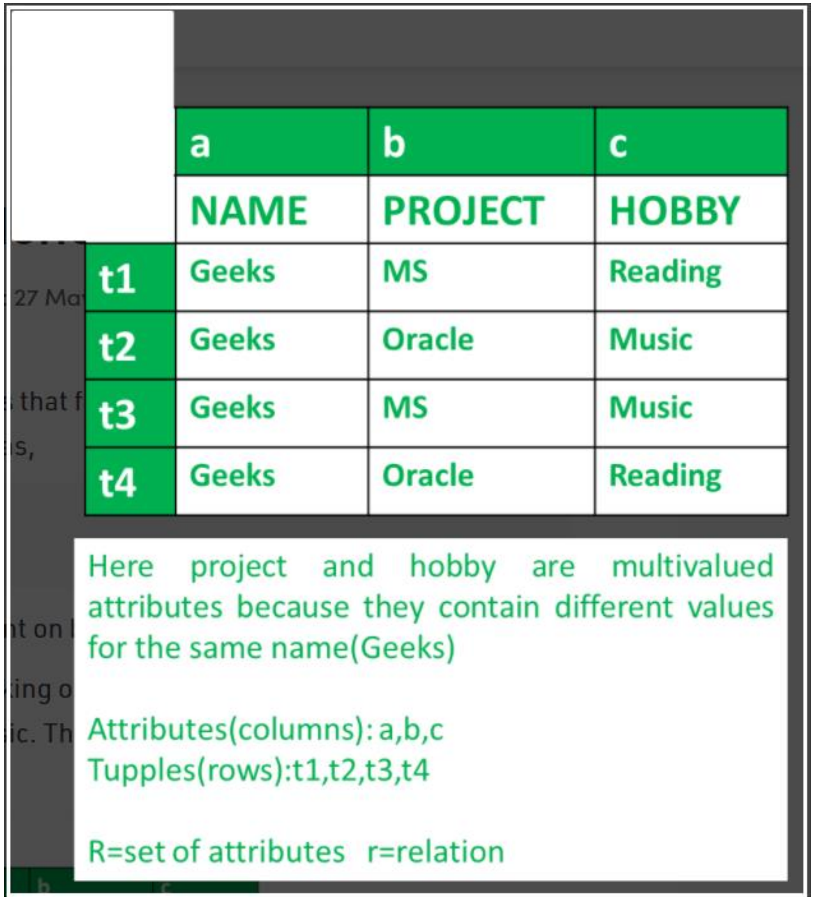
Customer

CustomerID		CustomerName	CustomerCity	Population
C1000		Ford	Dearborn	94000
C1010		GM	Detroit	670000
C1020		Dell	Austin	950000
C1030		HP	Palo Alto	67000
C1040		Apple	Cupertino	60000
C1050		Boeing	Chicago	2700000

- A table is in BCNF Normal Form if:
  - The table is in 3rd normal form, and
  - it has no more than one **Candidate** Key
    - A Candidate key is a unique key as the primary key to identify a record uniquely in a table but a table can have multiple candidate keys. A candidate key may or may not be a primary key.
- A table is in 4rth Normal Form if:
  - The table is in BCDF normal form, and
  - It should have no multi-valued dependency.

1NF, 2NF, 3NF, BCNF, 4NF

- MVD or multivalued dependency means that for a single value of attribute ‘a’ multiple values of attribute ‘b’ exist. We write it as,
  - a --> --> b



	a	b	c
	NAME	PROJECT	HOBBY
t1	Geeks	MS	Reading
t2	Geeks	Oracle	Music
t3	Geeks	MS	Music
t4	Geeks	Oracle	Reading

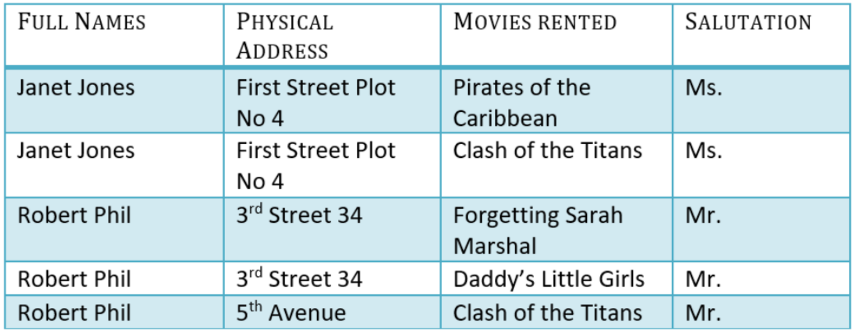
Here project and hobby are multivalued attributes because they contain different values for the same name(Geeks)

Attributes(columns): a,b,c
Tupples(rows):t1,t2,t3,t4

R=set of attributes r=relation

1NF, 2NF, 3NF, BCNF, 4NF

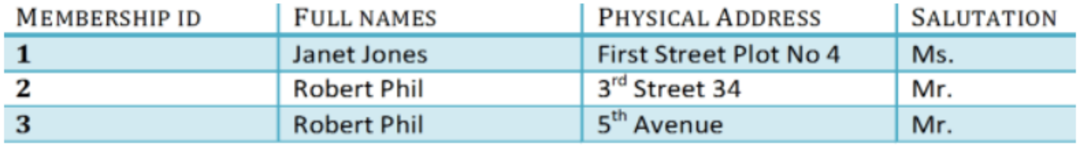
1NF Example



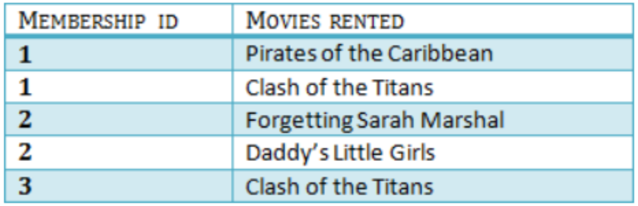
FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Example of 1NF in DBMS

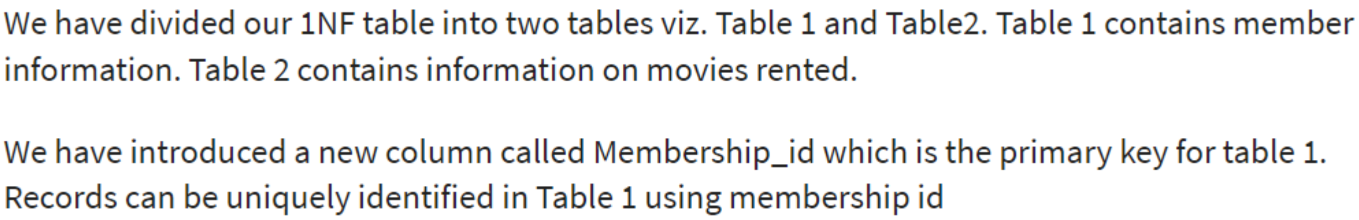
2NF Example



MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.



MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans



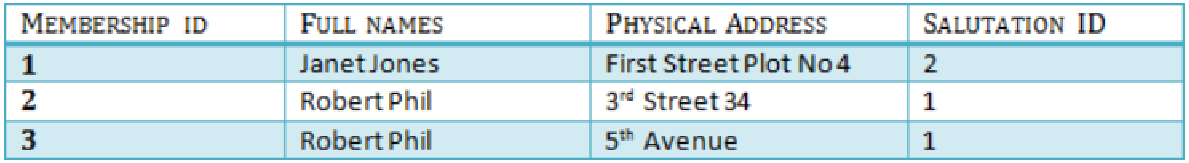
We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

We have introduced a new column called Membership_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

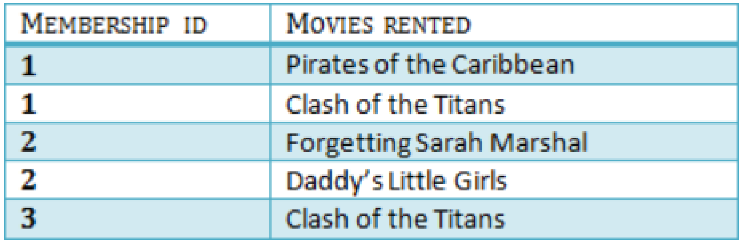
1NF, 2NF, 3NF, BCNF, 4NF

3NF Example

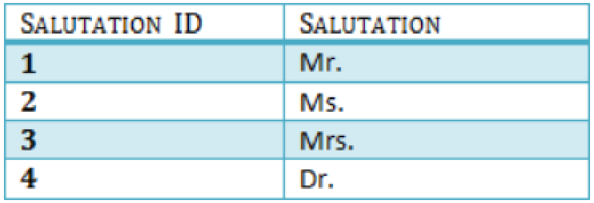
Below is a 3NF example in SQL database:



Membership ID	Full Names	Physical Address	Salutation ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1



Membership ID	Movies rented
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans



Salutation ID	Salutation
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

We have again divided our tables and created a new table which stores Salutations.

Complex Types and SQL

- Extensions introduced in SQL:1999 to support complex types:
  - Collection and large object types
    - **Nested relations** are an example of collection types
  - Structured types
    - **Nested record structures** like composite attributes
  - Inheritance
  - Object orientation
    - Including object identifiers and references
- Not fully implemented in any database system currently
  - But some features are present in each of the major commercial database systems
    - Read the manual of your database system to see what it supports

Structured Types and Inheritance in SQL

- **Structured types** (a.k.a. **user-defined types**) can be declared and used in SQL

```
create type Name as
     (firstname          varchar(20),
      lastname           varchar(20))
      final

create type Address as
     (street        varchar(20),
      city          varchar(20),
      zipcode      varchar(20))

      not final
```

  - Note: **final** and **not final** indicate whether subtypes can be created

- Structured types can be used to create tables with composite attributes

```
create table person (
     name        Name,
     address     Address,
     dateOfBirth date)
```

- **Dot notation** used to reference components: *name.firstname*

Structured Types (cont.)

- **User-defined row types**

```
create type CustomerType as (
    name Name,
    address Address,
    dateOfBirth date)
    not final
```

- Can then create a table whose rows are a user-defined type

```
create table customer of CustomerType
```

- Alternative using **unnamed row types (απροσδιόριστοι τύποι γραμμών)**.

```
create table person_r(
        name      row(firstname  varchar(20),
                      lastname   varchar(20)),
        address   row(street     varchar(20),
                      city       varchar(20),
                      zipcode    varchar(20)),
        dateOfBirth date)
```

Methods

- **create table** *customer* **of** *CustomerType*
- Can add a method declaration with a **structured type.**

```
method ageOnDate (onDate date)
        returns interval year;
```

- Method body is given separately.

```
create instance method ageOnDate (onDate date)
        returns interval year
        for CustomerType
begin
        return onDate - self.dateOfBirth;
end
```

type CustomerType

- We can now find the age of each customer:

```
select name.lastname, ageOnDate (current_date)
from customer
```

Table Customer of type CustomerType

Constructor Functions
Συναρτήσεις Δημιουργίας

- **Constructor functions** are used to create values of structured types
- E.g.
```
create function Name(firstname varchar(20), lastname varchar(20))
returns Name
begin
    set self.firstname = firstname;
    set self.lastname = lastname;
end
```
- To create a value of type *Name,* we use
```
   new Name(‘John’, ‘Smith’)
```
- Normally used in insert statements
```
insert into Person values
     (new Name(‘John’, ‘Smith),
      new Address(’20 Main St’, ‘New York’, ‘11001’),
      date ‘1960-8-22’);
```

Type Inheritance
Κληρονομικότητα Τύπων

- Suppose that we have the following type definition for people:

```
        create type Person
            (name varchar(20),
             address varchar(20))
```

- Using inheritance to define the student and teacher types

```
      create type Student
        under Person
        (degree        varchar(20),
         department  varchar(20))
      create type Teacher
        under Person
        (salary          integer,
         department  varchar(20))
```

- Subtypes can redefine methods by using **overriding method (επικάλυψη μεθόδου)** in place of **method** in the method declaration

Multiple Type Inheritance
Πολλαπλή Κληρονομικότητα

- SQL:1999 and SQL:2003 do not support multiple inheritance
- If our type system supports multiple inheritance, we can define a type for teaching assistant as follows:

```
create type Teaching Assistant
   under Student, Teacher
```

- To avoid a conflict between the two occurrences of *department* we can rename them

```
create type Teaching Assistant
 under
  Student  with (department as student_dept ),
  Teacher  with (department as teacher_dept )
```

- Each value must have a **most-specific type (πιο συγκεκριμένο τύπο)**

Table Inheritance
Κληρονομικότητα Πινάκων

- Tables created from subtypes can further be specified as **subtables**
- E.g. **create table** *people* **of** *Person;*
  create table *students* **of** *Student* **under** *people;*
  create table *teachers* **of** *Teacher* **under** *people;*
- Tuples added to a subtable are automatically visible to queries on the supertable
  - E.g. query on *people* also sees *students* and *teacher*s.
  - Similarly updates/deletes on *people* also result in updates/deletes on subtables
  - To override this behaviour, use "**only** *people"* in query
    - (eg select/update/delete….from only people……)
- ~~Conceptually, multiple inheritance is possible with tables~~
  - ~~e.g. *teaching_assistants* under *students* and *teachers*~~
  - ~~*But is not supported in SQL currently*~~
    - ~~So we cannot create a person (tuple in *people*) who is both a student and a teacher~~

Consistency Requirements for Subtables Απαιτήσεις Συνέπειας Υποπινάκων

- Consistency requirements on subtables (υποπίνακες) and supertables (υπερπίνακες – γονικοί πίνακες).
  - Each tuple of the supertable (e.g. *people)* can correspond to at most one tuple in each of the subtables (e.g. *students* and *teachers)*
    - ***Violation results to have two records in teachers or students for the same person***
  - Additional constraint in SQL:1999:

    All tuples corresponding to each other (that is, with the same values for inherited attributes) must be derived from one tuple (which is inserted into one table).
    - That is, each entity must have a most specific type
    - **Violation: To have a tuple in *people* corresponding to a tuple each in *students* and *teachers***

Array and Multiset Types in SQL
Τύποι Πινάκων και Πολλαπλών Συνόλων στην SQL

- Example of array and multiset declaration:

```
create type Publisher as
    (name            varchar(20),
     branch           varchar(20));
create type Book as
    (title             varchar(20),
     author_array   varchar(20) array [10],
     pub_date         date,
     publisher        Publisher,
     keyword-set    varchar(20) multiset);

 create table books of Book;
```

Creation of Collection Values
Δημιουργία Τιμών Συλλογών

- Array construction

```
array [‘Silberschatz’,`Korth’,`Sudarshan’]
```

- Multisets

```
multiset [‘computer’, ‘database’, ‘SQL’]
```

- To create a tuple of the type defined by the books relation:

```
    (‘Compilers’, array[`Smith’,`Jones’],
          new Publisher (`McGraw-Hill’,`New York’),
                multiset [`parsing’,`analysis’ ])
```

- To insert the preceding tuple into the relation books

```
insert into books
values
    (‘Compilers’, array[`Smith’,`Jones’],
          new Publisher (`McGraw-Hill’,`New York’),
          multiset [`parsing’,`analysis’ ]);
```

Querying Collection-Valued Attributes
Ερωτήματα για Ιδιότητες Συλλογών

- To find all books that have the word "database" as a keyword,

```
select title
from books
where 'database' in (unnest(keyword-set ))
```

- We can access individual elements of an array by using indices
  - E.g.: If we know that a particular book has three authors, we could write:

```
select author_array[1], author_array[2], author_array[3]
from books
where title = `Database System Concepts'
```

- To get a relation containing pairs of the form "title, author_name" for each book and each author of the book

```
select B.title, A.author
from books as B, unnest (B.author_array) as A (author )
```

- To retain ordering information we add a **with ordinality** clause

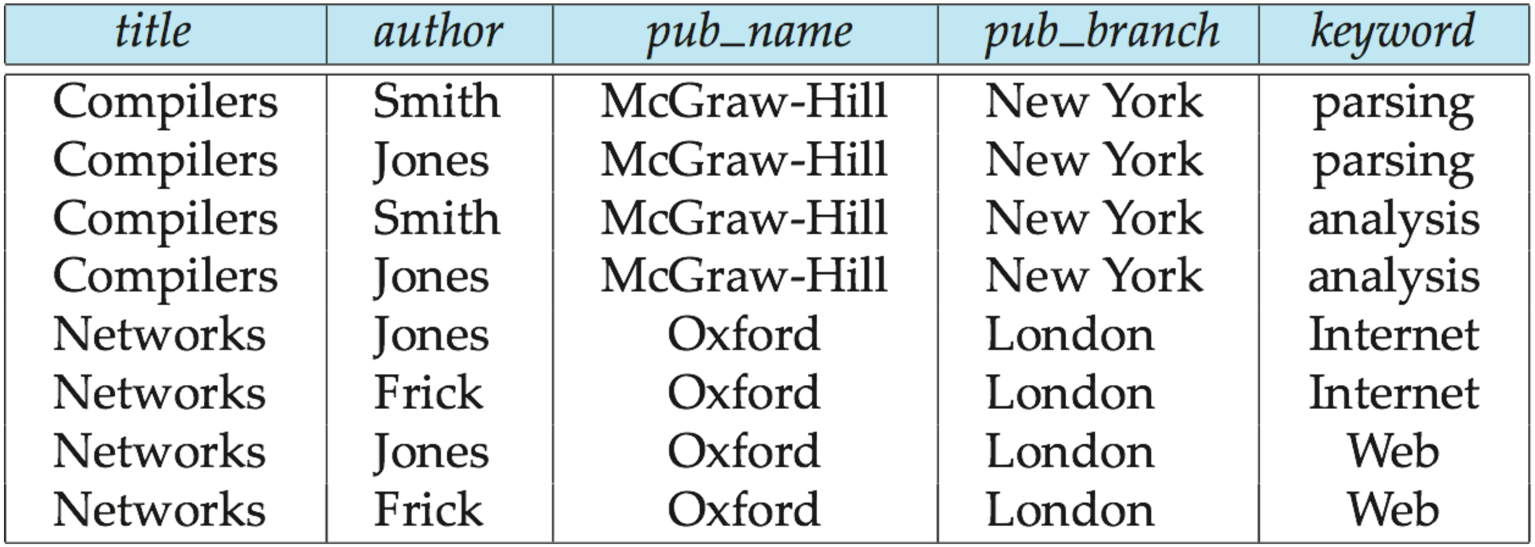
```
select B.title, A.author, A.position
from books as B, unnest (B.author_array) with ordinality as
          A (author, position )
```

Unnesting
Ακύρωση ένθεσης unnest

- The transformation of a nested relation into a form with fewer (or no) relation-valued attributes us called **unnesting**.
- E.g.

```
select title, A.author as author, publisher.name as pub_name,
             publisher.branch  as pub_branch, K.keyword
from books as B, unnest(B.author_array ) as A (author ),
             unnest (B.keyword_set ) as K (keyword )
```

- Result relation *flat_books*



title	*author*	*pub_name*	*pub_branch*	*keyword*
Compilers	Smith	McGraw-Hill	New York	parsing
Compilers	Jones	McGraw-Hill	New York	parsing
Compilers	Smith	McGraw-Hill	New York	analysis
Compilers	Jones	McGraw-Hill	New York	analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web

Nesting
Ένθεση

- **Nesting** is the opposite of unnesting, creating a collection-valued attribute
- Nesting can be done in a manner similar to aggregation, but using the function **colect**() in place of an aggregation operation, to create a multiset
- To nest the *flat_books* relation on the attribute *keyword*:

```
select title, author, Publisher (pub_name, pub_branch ) as publisher,
          collect (keyword)  as keyword_set
from flat_books
groupby title, author, publisher
```

- To nest on both authors and keywords:

```
 select title, collect (author ) as author_set,
           Publisher (pub_name, pub_branch) as publisher,
                      collect  (keyword ) as keyword_set
from   flat_books
group by title, publisher
```

Nesting (Cont.)

- Another approach to creating nested relations is to use subqueries in the **select** clause, starting from the 4NF relation *books4*

```
select title,
        array (select author
                 from authors as A
                 where A.title = B.title
                 order by A.position) as author_array,
        Publisher (pub-name, pub-branch) as publisher,
        multiset (select keyword
                    from keywords as K
                    where K.title = B.title) as keyword_set
from books4 as B
```

Object-Identity and Reference Types
Ταυτότητα Αντικειμένων κ Τύποι Αναφοράς

- Define a type *Department* with a field <u>*name*</u> and a field <u>*head*</u> which is a reference to the **type *Person***, with **table *people*** as scope:

  create type *Department* (
      name **varchar** (20),
      head **ref** (*Person*) **scope** *people*)

- We can then create a table *departments* as follows

  create table *departments* **of** *Department*

- ~~We can omit the declaration **scope** people **(πεδίο δράσης)** from the type declaration and instead make an addition to the **create table** statement:~~

  ~~**create table** *departments* **of** *Department*~~
  ~~(*head* **with options scope** *people*)~~

- Referenced table must have an attribute that stores the identifier, called the **self-referential attribute**

  create table *people* **of** *Person*
  ref is *person_id* **system generated**;

Initializing Reference-Typed Values

- To create a tuple with a reference value, we can first create the tuple with a null reference and then set the reference separately:

```
insert into departments
       values (`CS’, null)
update departments
     set head = (select p.person_id
                          from people as p
                            where name = `John’)
     where name = `CS’
```

User Generated Identifiers
Αναφορά Δημιουργούμενη από Χρήστη

- The type of the object-identifier must be specified as part of the type definition of the referenced table, and
- The table definition must specify that the reference is user generated

```
create type Person
    (name varchar(20)
     address varchar(20))
   ref using varchar(20)
create table people of Person
  ref is person_id user generated
```

- When creating a tuple, we must provide a unique value for the identifier:

```
insert into people (person_id, name, address ) values
   (‘01284567’, ‘John’, `23 Coyote Run’)
```

- We can then use the identifier value when inserting a tuple into *departments*
  - Avoids need for a separate query to retrieve the identifier:

```
insert into departments
values(`CS’, `02184567’)
```

User Generated Identifiers (Cont.)

- Can use an existing primary key value as the identifier:

```
create type Person
      (name varchar (20) primary key,
       address varchar(20))
    ref from (name)
create table people of Person
    ref is person_id derived
```

- When inserting a tuple for *departments*, we can then use

```
insert into departments
    values(`CS’,`John’)
```

Path Expressions

- Find the names and addresses of the heads of all departments:

  select *head* −>*name*, *head* −>*address*
  from *departments*

- An expression such as “head−>name” is called a **path expression**
- Path expressions help avoid explicit joins
  - If department head were not a reference, a join of *departments* with *people* would be required to get at the address
  - Makes expressing the query much easier for the user

Implementing O-R Features
Υλοποίηση Αντικειμενο-Σχεσιακών Λειτουργιών

- Similar to how E-R features are mapped onto relation schemas
- Subtable implementation
  - Each table stores primary key and those attributes defined in that table

  or,

  - Each table stores both locally defined and inherited attributes