**Object-Relational Data Models** make traditional relational data model better by implementing object-oriented principles and constructs to handle complex data types, to enhance their capabilities to accommodate complex, non-atomic values & nested relations.

**Complex Data Types:** It addresses the need to handle non-atomic domains, which are indivisible entities. Examples: sets of integers, strings, or tuples, which expand the types of things we can put into our db to make it more useful for handling complicated information.

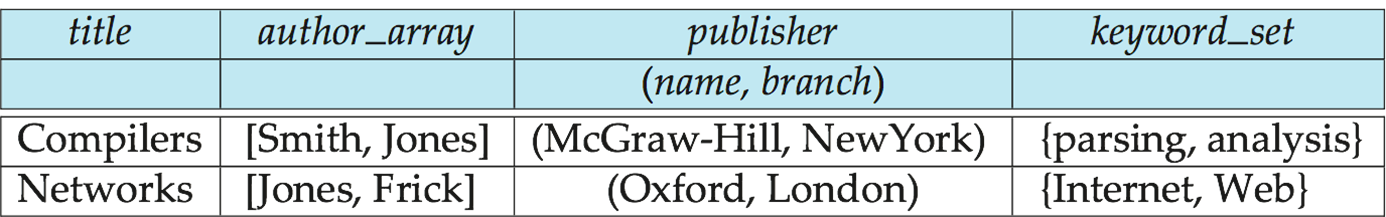
**Preservation of Relational Foundations:** Even though we add object-oriented ideas, the model still sticks to the main ideas of relational databases. One big idea is declarative access. This means users can ask for data without worrying about how it's stored. It keeps things simple and flexible for everyone using the database.

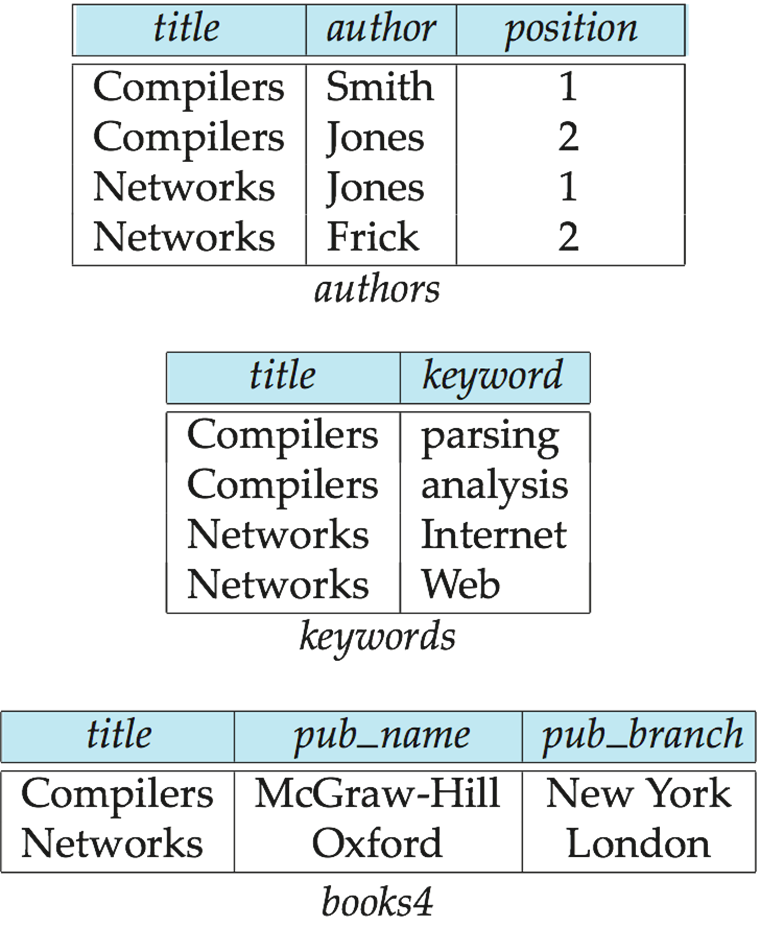
**Upward Compatibility:** Ensure that applications and systems built on previous relational dbs can easily switch to the updated model without problems. Ensures that existing queries, data structures, and applications remain functional and can be adapted to use the improved ways of modeling provided by the object-relational approach.

**Features of Object-based DBMS:-**

* *Complex Data Support:* Allow management of complex DS, including nested objects and non-atomic domains, enabling more flexible and natural data modeling.
* *Inheritance:* Support inheritance hierarchy (code reuse & better data organization).
* *Encapsulation:* Encapsulates data and behavior within objects, promoting data integrity (accuracy) and security by controlling access to data.
* *Methods and Functions:* They enable the definition and execution of methods and functions with objects, which helps you do more complex things with your data.
* *User-Defined Data Types:* Permit user-defined data types, providing greater flexibility in data representation and manipulation.

**Example of a Nested Relation:**

Consider a library information system.

This structure creates a nested relation as it doesn't violate 1st normal form (1NF) (each attb value should be atomic).

*4NF Decomposition of Nested Relation:-*

Assume each book's title as unique identifier, although in the real world, ISBN can be unique identifier. To convert nested relation into 4NF use the following schemas:

1. (title, author, position)

2. (title, keyword)

3. (title, pub-name, pub-branch)

4NF design requires users to include joins in their queries.

**Structured types,** also known as user-defined types. They are used to define custom data structures, enhancing the flexibility and organization of data in a database.

1. *Defining Structured Types:* declared using the CREATE TYPE statement. FINAL= subtypes cannot be created for this type. NOT FINAL allows for the creation of subtypes.

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;

1. *Using Structured Types in Tables:* Define composite (can be subdivided) attributes in tables. Dot notation is used to reference components of structured types, such as name.firstname.

CREATE TABLE person (name Name, address Address, dateOfBirth DATE);

1. *User-Defined Row Types for Table Creation:* This creates a user-defined row type named PersonType.

CREATE TYPE PersonType AS (name Name, address Address, dateOfBirth DATE); NOT FINAL;

1. *Tables can then be created using this type:*

CREATE TABLE person OF PersonType;

1. *Alternative Using Unnamed Row Types:*

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** are blocks of code that perform specific actions or tasks.

1. *Adding a Method Declaration with a Structured Type:* This declaration defines a method named ageOnDate which takes a DATE parameter and returns an INTERVAL YEAR.

CREATE TYPE PersonType AS (  
 name Name, address Address,  
 dateOfBirth DATE, METHOD ageOnDate (onDate DATE)

RETURNS INTERVAL YEAR);

1. *Creating an Instance Method for a Structured Type:* This instance method calculates the age of a person based on the provided date (onDate) and their dateOfBirth.

CREATE INSTANCE METHOD ageOnDate (onDate DATE) RETURNS INTERVAL YEAR

FOR PersonType

BEGIN

RETURN onDate - self.dateOfBirth;

END;

1. *Querying Using Structured Types and Methods:*

SELECT name.lastname, ageOnDate(CURRENT\_DATE) AS age FROM person;

1. *Using Functions with Composite Types:*

CREATE TYPE BookType AS (

book\_id SERIAL, title VARCHAR(100),

author VARCHAR(50), publication\_year INTEGER

);

CREATE FUNCTION display\_book\_info(book BookType)

RETURNS VARCHAR AS $$

BEGIN

RETURN 'Book ID: ' || book.book\_id ||

', Title: ' || book.title ||

', Author: ' || book.author ||

', Publication Year: ' || book.publication\_year;

END;

$$ LANGUAGE plpgsql;

CREATE TABLE Library (

book\_data BookType

);

INSERT INTO Library VALUES (

ROW(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925)

);

SELECT display\_book\_info(book\_data) AS book\_info FROM Library;

**Constructor functions** are commonly used to instantiate objects with predefined properties and behaviors.

CREATE FUNCTION Name(firstname VARCHAR(20), lastname VARCHAR(20))

RETURNS Name

BEGIN

SET self.firstname = firstname;

SET self.lastname = lastname;

END;  
  
NEW keyword is used to create a new instance:

NEW Name('John', 'Smith');

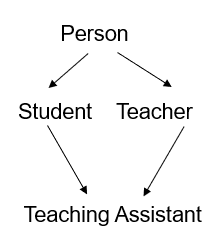
INSERT INTO Person VALUES (

NEW Name('John', 'Smith'),

NEW Address('20 Main St', 'New York', '11001'),

DATE '1960-08-22'

);



**Type Inheritance:**

CREATE TYPE Person  
 (name VARCHAR(20),  
 address VARCHAR(20))

CREATE TYPE Student UNDER Person (

degree VARCHAR(20),

department VARCHAR(20)

);

CREATE TYPE Teacher UNDER Person (

salary INTEGER,

department VARCHAR(20)

);

Multiple Type Inheritance:

If our type system supports multiple inheritance, we can define a type for teaching assistant as follows:  
CREATE TYPE TeachingAssistant UNDER Student, Teacher;

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

CREATE TYPE TeachingAssistant UNDER

Student WITH (department AS student\_dept),

Teacher WITH (department AS teacher\_dept);

**Table inheritance** refers to the ability to create tables that are specialized versions (subtypes) of more general tables (supertypes).

1. *Creating Subtables:* from existing tables, forming a hierarchy. In this example, students and teachers are subtables of the more general people table.

CREATE TABLE people OF Person;

CREATE TABLE students OF Student UNDER people;

CREATE TABLE teachers OF Teacher UNDER people;

1. *Visibility and Updates:* Tuples added to a subtable are automatically visible to queries on the supertable. Updates or deletes on the supertable result in corresponding updates or deletes on the subtables. For example, queries on people also see data from students and teachers.
2. *Multiple Inheritance:* Conceptually, multiple inheritance is possible with tables. This creates a table for teaching assistants under both the students and teachers tables.

CREATE TABLE teaching\_assistants UNDER students, teachers;

**Array and Multiset Types in SQL:** This example demonstrates the declaration of array and multiset types within SQL for the Book type.

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;

**Collection-valued attributes:-**

INSERT INTO books

VALUES ('Compilers', ARRAY['Smith', 'Jones'], NEW Publisher('McGraw-Hill', 'New York'), MULTISET['parsing', 'analysis']);

SELECT title

FROM books

WHERE 'database' IN (UNNEST(keyword\_set));

SELECT author\_array[1], author\_array[2], author\_array[3]

FROM books

WHERE title = 'Database System Concepts';

CREATE TABLE Person (

PersonID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

PhoneNumbers VARCHAR(10)[],

EmailAddresses MULTISET(TEXT)

);

**Nesting** involves creating a collection-valued attribute by aggregating data using the COLLECT() function.

SELECT title, author, Publisher(pub\_name, pub\_branch) AS publisher, COLLECT(keyword) AS keyword\_set FROM flat\_books

GROUP BY title, author, publisher;

**Nesting on Multiple Attributes:**

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;

This query nests on both authors and keywords, creating sets of authors and keywords for each title and publisher combination.

**Using Subqueries for Nesting:** Another approach to creating nested relations is using subqueries in the SELECT clause.

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-Relational Mapping (ORM):**

* ORM is like a translator between your computer code (objects) and the database (relations).
* Objects are transient without permanent object identity.
* It helps fetch data from the database and convert it into objects that programs can use. Similarly, it also helps save changes made to objects back into the database.
* Popular ORM systems like Hibernate provide APIs for transaction management, object retrieval, and more.
* However, ORM systems may have overheads, especially for bulk updates.

**Comparison of Object-Oriented (OO) and Object-Relational (OR) Databases:**

* *Relational Systems:* Relational systems offer simple data types, powerful query languages, and high protection mechanisms.
* *Persistent-Programming-Language-Based Object-Oriented Databases (OODBs):* OODBs support complex data types, integration with programming languages, and high performance.
* *Object-Relational Systems:* OR systems provide complex data types, powerful query languages, and high protection mechanisms.
* *Object-Relational Mapping (ORM) Systems:* ORM systems offer complex data types integrated with programming languages, operating as a layer on top of relational database systems.