Data and Programming

SQL, functions, ORM and ER

Author: Tomasz Gil – consultant at Launch Consulting

Introduction

In recent years we have seen an increased interest in functional programming and a move away from focus on object orientation.

While object orientation is an organization of information where elements of state are combined with operations, the functional approach emphasizes values.

Values are the center of information and they are consumed and produced by functions.

Object-orientation and functional (value-centric) orientation are presently somewhat at a standoff, and this article aims to clarify the position of each side and propose the most appropriate competence area for each side.

The idea of this article are inspired by recent presentations and papers such as:

- Sam Roberton -YOW! Lambda Jam 2019 -Functional Programming in ... SQL? https://www.youtube.com/watch?v=ZbJbnxQ6GiI
- Rich Hickey –2012 - The Value of Values with Rich Hickey:https://www.youtube.com/watch?v=-6BsiVyC1kM
- CJ Date -"SQL and Relational Theory" - book
- Post on jooq.com blog arguing against ORM

The world of data is the world of values - reflecting the real world (to the degree of interest to the application at hand):

Data has:

- Structure: attributes, references, collections
- Values: magnitudes, names, identifiers
- Mutability: because the world changes

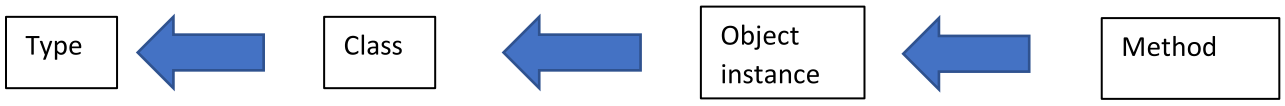
Values are the primitive building blocks of data (as in primitive values) and are primarily

- magnitudes - which can be compared to others and measured
- names - which are whatever they need to be

- identifiers - values that have meaning in construction of references

Data structure leads readily to object orientation and converts structures into entities:

- Type is a declaration of attributes and operations (methods)
- Class is implementation of a Type
- Object is instance of a Class
- Method is invoked on an instance



Object instance is referred to as entity and it permits mutations of values by designated methods.

Type is the public definition of specific classes of entities - separated from the primitive value content and from operations.

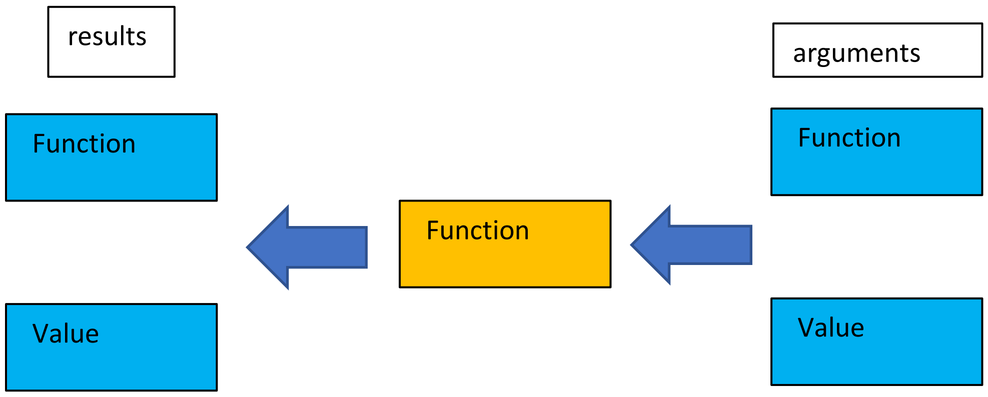
Functional programming and functions are around transformations of values.

Functions transform values (and other functions):

x,y,z,a,b,c - values

f,g,h - functions

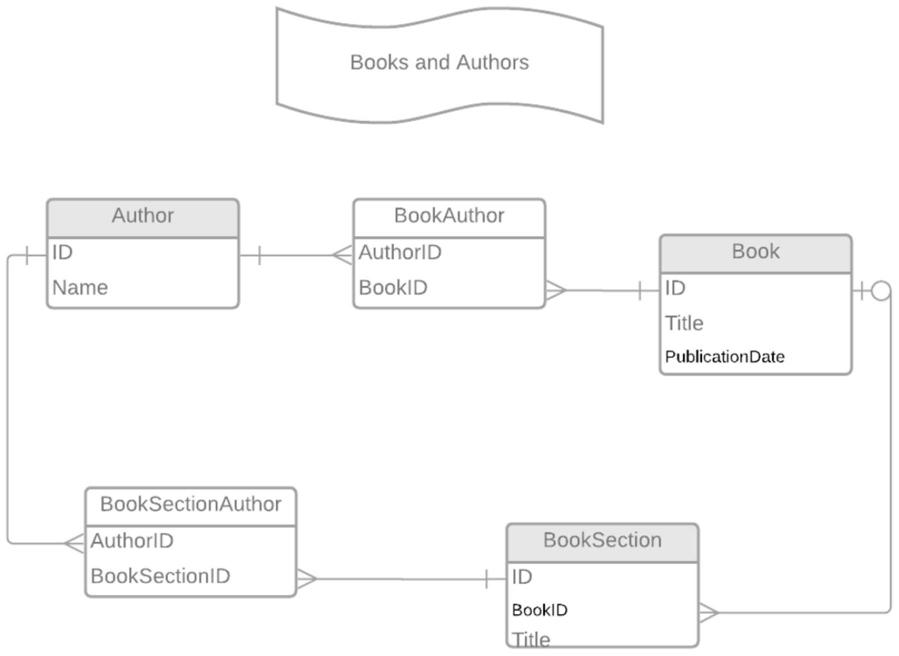
(x,y,z,g,…) <- f(a,b,c,h,…)



A function takes values and other functions as arguments

and returns new values and new functions.

Let us look at an example data model - Book, Author and BookSection



It shows that a Book can have many Sections as well as many Authors. Authors can also be associated with multiple Books and BookSections.

Here is how this data model will be seen as Entities:

```
Class Book {

      Int ID

      String Title

      Date PubDate

      List<BookSection> sections


      AddSection(params...)   // a method that can be on a Service level
      class such as BookManager

}


Class BookSection {

      Int ID
      String Title
      Book book
}
```

Book and BookSection as Relations look quite different.

Relations are subsets of cross-products of different domains. This is non-intuitive and requires some theoretical introduction which follows below. Relations are about tying together values from suitable domains to represent world-level entities. Some of these values have significance only in tying together two relations – this referential significance is apart from the substantive significance of attribute values such as book title, book section title and book publication date.

In current database technologies relations occur as tables or table-valued data. Relation is really an extension of the notion of value.

Book

Integer ID	string Title	Date PublicationDate

BookSection

Integer ID	string Title	Integer BookID

Entity (ER) vs Relational approach can be illustrated in this Month sequence example.

This is a simple example showing how each of our modes represents information

```
Class MonthSequence{
Int seq,
String monthName }
Instance1 = new MonthSequence(3, "March")
Instance2 = new MonthSequence(1, "January")
Instance3 = new MonthSequence(5, "May")
```

…

vs

3	**March**
1	**January**
5	**May**
…	

In the relational approach all the Entities are grouped together in a Relation making it apparent that there are possible conditions of uniqueness, possible limits on the domain of values in each column. In the Entity model the bits of information are separated into individual instances sharing structure.

Cross product of domains – what is it?

Domain is a set of values of a certain (primitive) type suitable for a given representation.

The month names are not just any strings - but only the strings that are names of months, and only the numbers 1-12 would be the month numbers.

Let's take 3 domains - where values are a letter of the alphabet concatenated with a number:

D1: [a-z]1 – meaning any letter followed by number 1

D2: [a-z]2 - ...

D3: [a-z]3

Cross product D1xD2xD3 means all possible tuples - such as:

D1	D2	D3
a1	b2	d3
a1	o2	m3
b1	m2	x3

There would be 26^3 of different tuples - or generally *card*(D1)**card*(D2)**card*(D3) - with *card* standing for cardinality of a domain.

The position of the domain in the cross product can be seen as an attribute designation constituting a definition of entity while the tuples would be object instances. The same domain can occur at different positions and play the role of a different attribute. For example, the domain of *dates* would be used on a *startDate* as well as *endDate* attribute.

A relation is a *subset* of the cross product. Being a *subset* adds meaning to the construct by including and excluding its members. Relational data has meaning because certain tuples are included or excluded in specific meaningful relations rather than arranged into a structure. This happens in the actual tables when certain rows are inserted/updated to reflect the true state of the world – and it happens in table-valued relations that are produced as results of queries – because the queries impose conditions of interest to the user of the data.

Operations on table valued variables

Let's use this simple data model – Doctor,Patient, Appointment

- Doctor: id, name
- Patient: id, name
- Appointment: doctor_id, patient_id, appt_time

Appointment

DoctorID	PatientID	Appointment_time
1	2	11am
2	11	3pm
2	1	2pm
3	5	6pm

Doctor

DoctorID	Doctor_Name
2	Don
3	Martin
1	Natalie

Patient

PatientID	Patient_Name
2	Colleen
11	Max
1	Arthur
4	Benji

The basic operations on table-valued variables:

- composition - FROM a JOIN b ON b.id = a.id

- filtering – WHERE b.date > '2021-01-01'
- projection – SELECT a.name, b.date, sum(b.amount)
- aggregation – GROUP BY a.shipDate

are now represented in diagrams.

Composition:

Patient_name	Doctor_id	id	Patient_id	Appt_time
Arthur	2	1	1	2pm
Colleen	1	2	2	11am
Max	2	11	11	3pm

Patient_name	id
Colleen	2
Arthur	1
Benji	4
Max	11

Patient_id	Appt_time	Doctor_id
2	11am	1
1	2pm	2
11	3pm	2
5	6pm	3

Filtering:

Patient_name	Doctor_id	Patient_id	Appt_time
Arthur	2	1	2pm
Max	2	11	3pm

Where doctor_id = 2

Patient_name	Doctor_id	Patient_id	Appt_time
Arthur	2	1	2pm
Colleen	1	2	11am
Max	2	11	3pm

Projection:

Patient_name	Appt_time
Arthur	2pm
Max	3pm
Colleen	11am

Select patient_name, Appt_time

Patient_name	Doctor_id	Patient_id	Appt_time
Arthur	2	1	2pm
Colleen	1	2	11am
Max	2	11	3pm

Aggregation:

Min_time	Doctor_id
6pm	3
11am	1
2pm	2

Min(appt_time) group by doctor_id

Patient_id	Appt_time	Doctor_id
2	11am	1
1	2pm	2
11	3pm	2
4	6pm	3

Observations on relational operations:

- No special entities participate in operations - just values organized in relations
- "Types" are more types of operations rather than types of data
- Logic resides in data – in presence/absence of tuples in relations - and in matching data values with themselves and with parameter values
- Queries are pure functions – insofar as the data can be seen as constant between mutations
- Programming is declarative – no loops – they are in the engine!

Two data perspectives

We have two views in which values appear reflecting the structure of the world – Entity-based and Relational view. The Entity (or Object) view best supports object-relational mapping (ORM) and entity-relational view where entities enter into relations with each other. The Entity-based view is well suited to support data mutation through application of object-based business logic. The Relational view is best suited to querying data in their complex relations. As it handles data in the form of table-valued items called relations it makes this form most amenable to serialization – that is external representation and transfer.

Compare worlds

Object world features	**Relational world features**
Business/Entity objects Data mutation Data structures Mapping	DTO objects Querying Aggregation Serialization

Comparing advantages

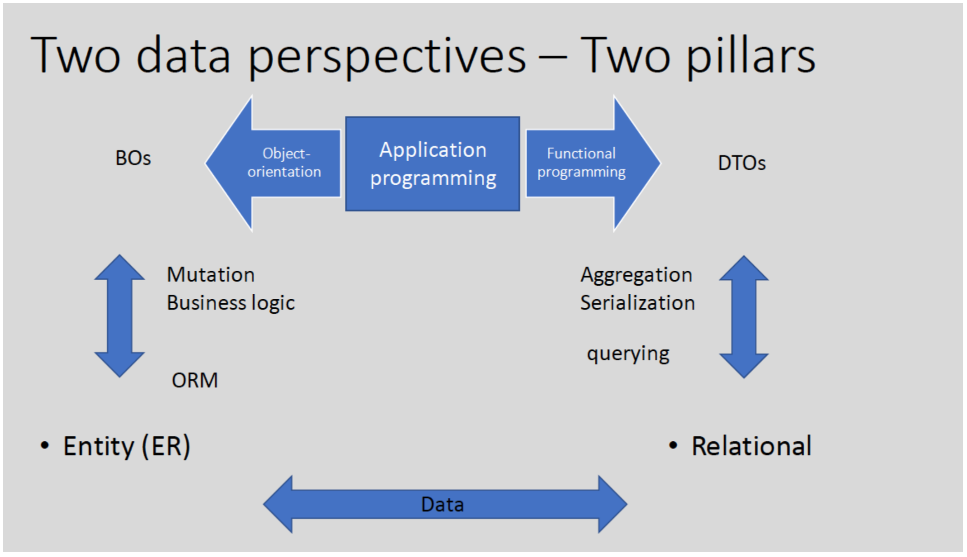
Object world boasts	**Relational world boasts**
logical organization around business logic combining methods with attributes - incl. mutating methods	table is never circular relationships are represented as joins queries are definitions of functions based on table data queries are functions returning tables SQL has subqueries, CTEs and nested queries - that really pass functions as arguments and chain query executions together

Comparing disadvantages

Object world lacks	**Relational world lacks**

termination of referential circularity enforcement of mutability conditions, by transactional and concurrency control	aggregation into objects objects as database types generalization of relational concept w/o tying it to physical storage or SQL constructs syntactically better language than SQL DML consistency

Summary in diagram



Conclusions

Relational world offers a foundation for functional programming

Objects/Entities are - collections of value attributes and other object references -easily relate to business features, mutability defined by business methods and their contract

Relations/Tables are - concise (normalized) expressions of relationships between values -direct business meaning not very clear, no definition or contract of mutability

JSON is not so much an object notation as notation for results of complex relational operations, such as aggregation and composition