"Category theory apparently is about reduction and finding patterns in patterns." – (My paraphrasing)
Discussing Sustainable Software Development with John (A) De Goes
https://www.youtube.com/watch?v=7hCVgoY4jjs

There are 3 big ideas in Category Theory:

Category
Functor
Natural Transformation

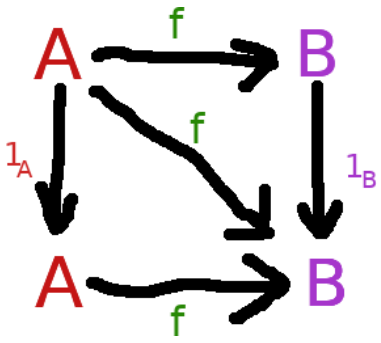
Definition of a Category:

+ A category C is a collection of objects and morphisms.
+ All objects must contain identity morphisms.
+ If for a pair of objects A and B, and a second pair of objects B and D in Category C there exists morphisms A → B and B → D, and B is the same object, then there must exist a composition A → D. Let f be the morphism from A → B, and g be the morphism from B → D. Then the composition morphism from A → D is g o f (g after f).

Additionally, identity and associativity laws must hold.

Identity:

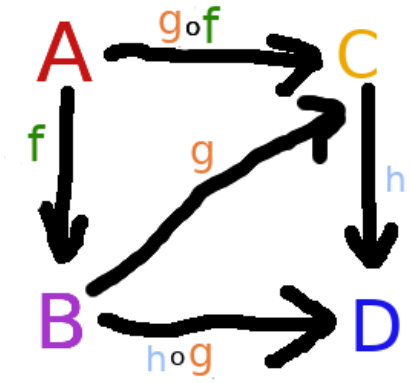
f o 1a = f = 1b o f



[2]

Associativity:

If morphism A → B is f, B → C is g , C → D is h then A → D is (h o g) o f = h o (g o f) = h o g o f



[2]

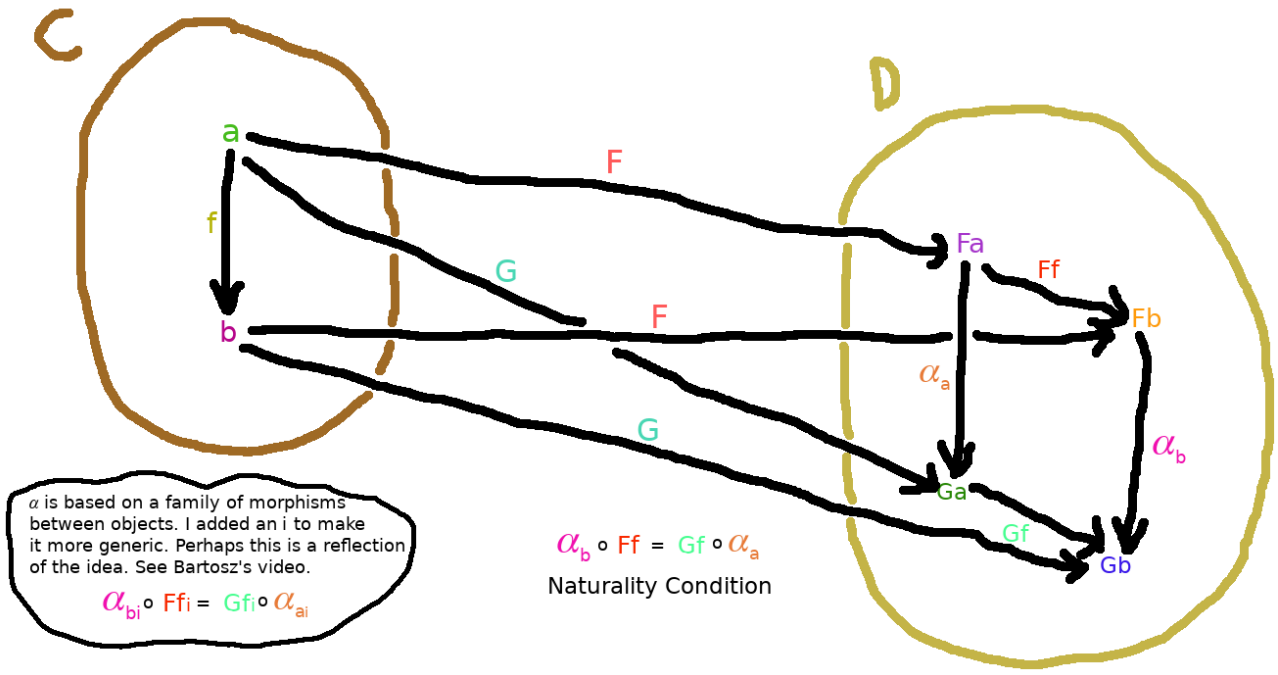
Note: A → B (has domain A, and codomain B)
In the category set, A = {a1,a2,a3,…,an} , B = {b1,b2,b3,…,bn}
A → B can represent maps between a1 → b1 , a2 → b3, a3 → b3, …, …. However each element in the domain can only map to one element in the codomain. (i.e a1 → b1 , a1 → b2, is not permissible)

The domain and codomain reverse for B → A. Category Theory stays at the object not element level. Focus stays on the morphisms between objects. Codrington expands on the concepts of maps, domain, and codomain for elements and how they trickle down from the concept of morphisms between objects.

Definition of a Functor:

"A Functor is a mapping between categories." [3] For two categories M and K, a functor maps objects from M to K. A functor preserves morphisms between objects when mapping to another category.

Definition of a Natural Transformation:



A possible relation between families of morphisms in two categories. Based on the diagram in [4] .

[1] Sergei Winitzki, The Science of Functional Programming , 2018 – 2020, Lulu
[2] Dr. Martin J.M. Codrington, Category Theory : The Beginnner's Introduction
https://www.youtube.com/playlist?list=PLm_IBvOSjN4zthQSQ_Xt6gyZJZZAPoQ6v
[3] Bartosz Milewski, Category Theory for Programmers, version 1.3.0-0-g6bb0bc0, August 12, 2019
[4] Bartosz Milewski, Category Theory 9.1: Natural transformations
https://www.youtube.com/watch?v=2LJC-XD5Ffo

Functorial Data Migration [5] and FQL [6] developed by Dr. David Spivak and Dr. Ryan Wisnesky utilizes Category Theory in its development. (**Please note that mention of their names does not imply an endorsement.** Mention here is to pose the question as to whether their work would be useful to the Self-Sovreign Identity Community.)

If schemas are represented as categories, a “constraint respecting mapping” F from schema S to Schema T may be constructed. Nodes in S map to nodes in T. Edges in S map to paths in T. [7] Three data migration functors: (sigma) Σ_F, (delta) Δ_F , and (pi) π_F arise from this.

$F: C \rightarrow D$ | $\Sigma_F = C_{inst} \rightarrow D_{inst}$ (left pushforward functor like union in sql)

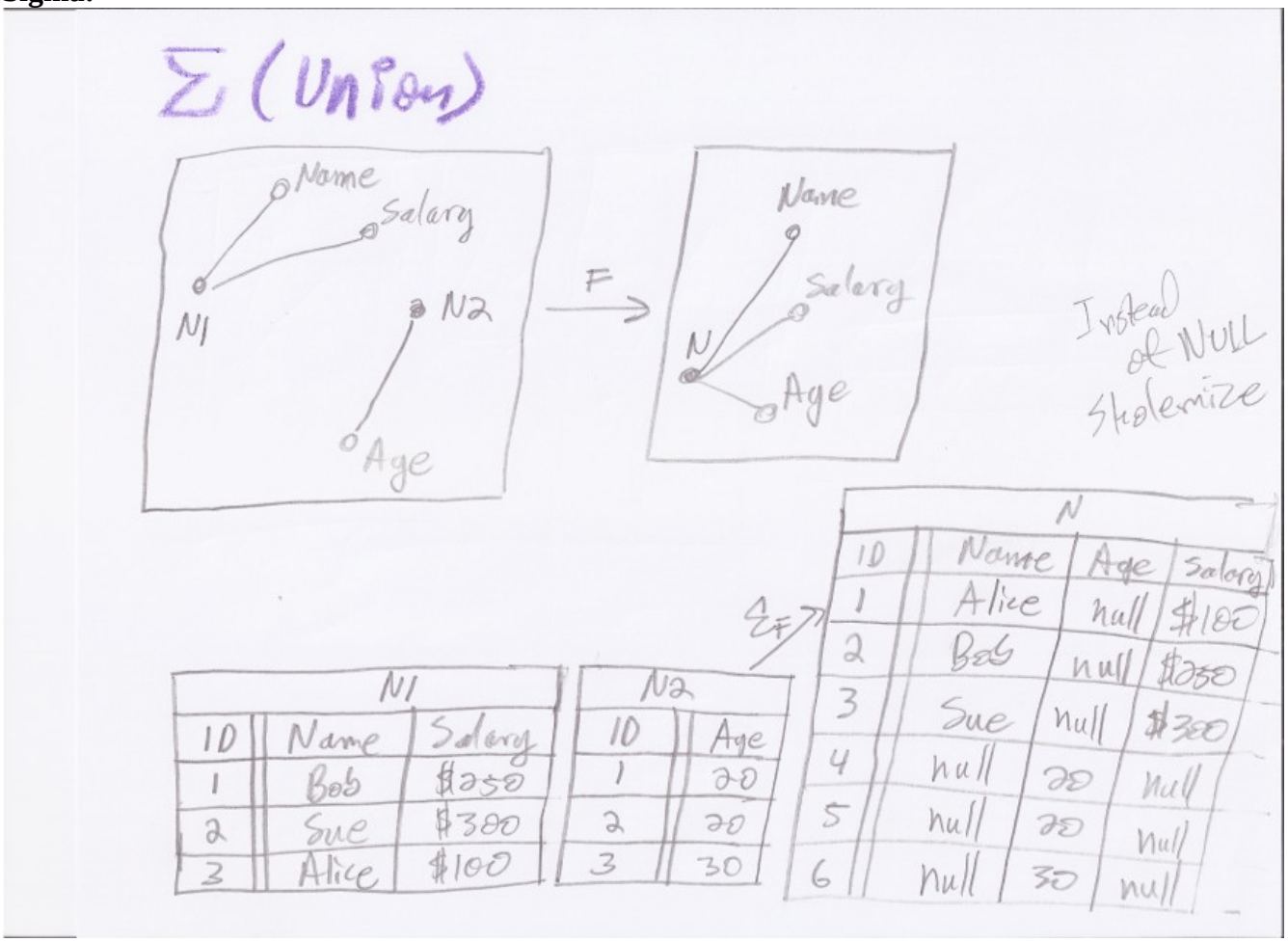
$F: C \rightarrow D$ | $\Delta_F = D_{inst} \rightarrow C_{inst}$ (pullback functor like projection in sql)

$F: C \rightarrow D$ | $\Pi_F = C_{inst} \rightarrow D_{inst}$ (right pushforward functor like join in sql)

SQL Tables to RDF with Grothendieck Construction . The FQL tool allowed for JSON import?

FQL Tool: https://github.com/CategoricalData/FQL

Sigma:



sketch from https://www.youtube.com/watch?v=Q0m8baqBrk4 (Ryan Wisnesky - A Functorial Query Language)

Sigma in FQL (guess):

```
schema C = {
 nodes
     N1,
     N2;
 attributes
        name0 : N1 -> string,
        salary0 : N1 -> string,
        age0 : N2 -> int;
  arrows;
  equations;
}

schema D = {
  nodes
        N;
  attributes
         name : N -> string,
         salary : N -> string,
         age : N -> int;
  arrows;
  equations;
}

mapping F = {
 nodes
   N1 -> N,
   N2 -> N;
 attributes
   name0 -> name,
   salary0 -> salary,
   age0 -> age;
 arrows;
} : C -> D


// this is for the sigma and pi functors
instance J = {
  nodes
        N1 -> {1,2,3},
        N2 -> {1,2,3};
  attributes
        name0 -> {(1,"Bob"),(2,"Sue"),(3,"Alice")},
```

```
        salary0 -> {(1,"$250"),(2,"$300"),(3,"$100")},
        age0 -> {(1,20),(2,20),(3,30)};
  arrows;
} : C

// this is is for the sigma functor
instance L = sigma F J

instance Q = delta F L

transform monad_unit = Q.return

instance M = sigma F Q

transform monad_counit = M.coreturn

instance H = delta F M

transform monad_join = delta H Q monad_counit
```

Presently this code throws the error.
Error in Instance L “not union compatible” For more see:
https://github.com/CategoricalData/FQL/search?q=not+union+compatible
“For Σ F to implementable in SQL, the functor F must satisfy the special condition of being a discrete op-fibration, which basically means “union compatible in the sense of Codd”. Mathematically, it is possible to define Σ for any schema mapping, not just mappings that are union-compatible. Such an unrestricted sigma is known as a “Left-hand” extension. The FQL IDE can compute such “unrestricted” sigmas, indicated by the keyword “SIGMA” (all caps). See the “full sigma” example for details.” – pg 15, fqlmanual in FQL repository

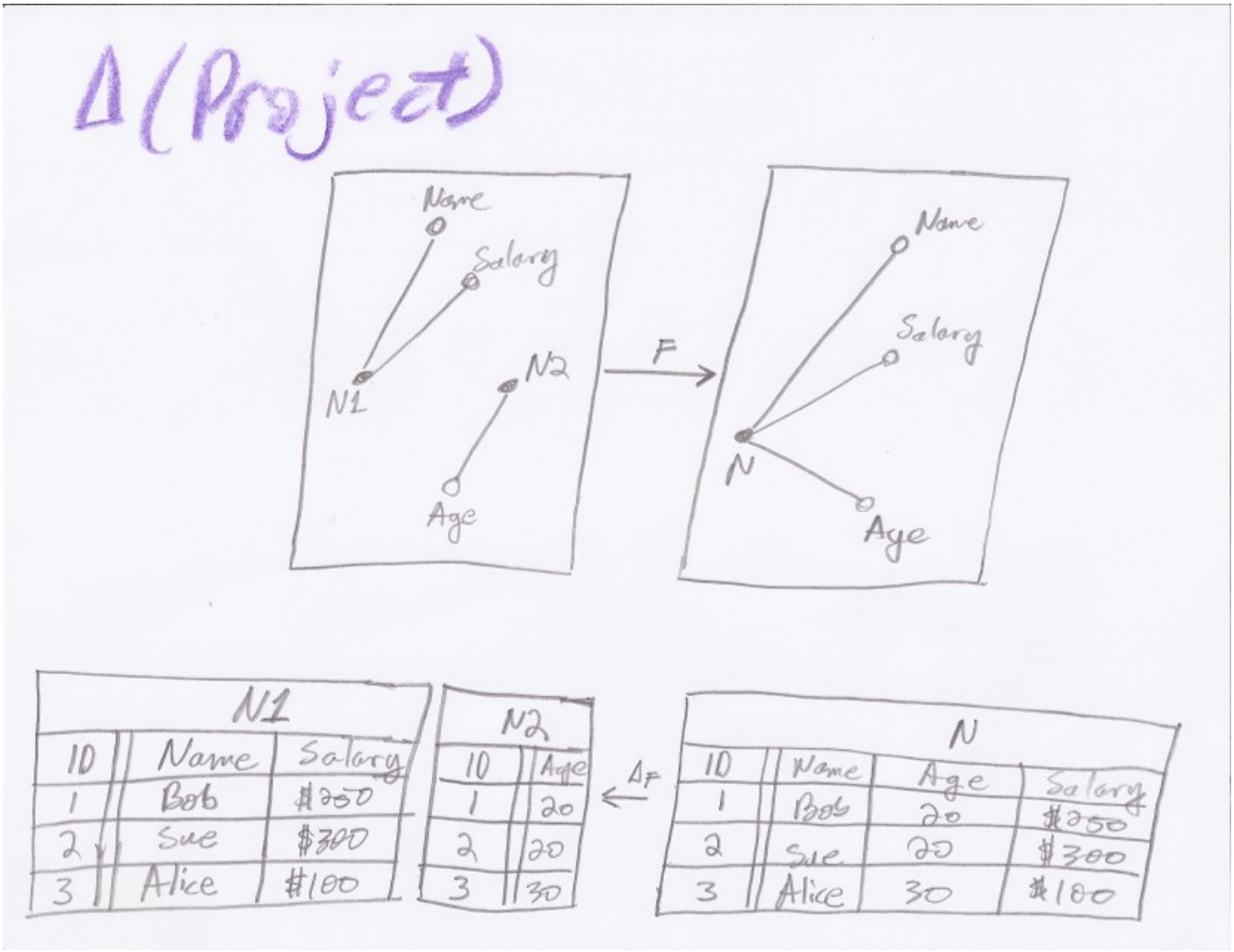
Interesting: “The FQL SQL compiler only supports sigma for mappings
that are discrete op-fibrations. However, we have added
support for full Sigma directly to the FQL IDE.” - Full Sigma Example in FQL

Maybe I am forgetting:

“We now have the
desired connection between database schemas and categories: Tables in a schema
are specified by vertices (or as we have drawn them in Diagram (1), by boxes);
columns are specified by arrows; and functional equivalence of foreign key paths
are specified by the category-theoretic notion of path equivalence (indicated by the
≃ symbol).**” – Functorial Data Migration by Spivak**

So are tables nodes, columns arrows, and equivalencies of foreign key paths equations?

Delta:



sketch from https://www.youtube.com/watch?v=Q0m8baqBrk4 (Ryan Wisnesky - A Functorial Query Language)

Delta in FQL (guess):

```
schema C = {
 nodes
     N1,
     N2;
 attributes
        name0 : N1 -> string,
        salary0 : N1 -> string,
        age0 : N2 -> string;
  arrows;
  equations;
}

schema D = {
  nodes
        N;
  attributes
         name : N -> string,
         salary : N -> string,
         age : N -> string;
  arrows;
  equations;
}

mapping F = {
 nodes
   N1 -> N,
   N2 -> N;
 attributes
   name0 -> name,
   salary0 -> salary,
   age0 -> age;
 arrows;
} : C -> D

// This is for the delta functor
instance K = {
  nodes
       N -> {1,2,3};
  attributes
       name -> {(1,"Bob"),(2,"Sue"),(3,"Alice")},
       salary -> {(1,20),(2,20),(3,30)},
       age -> {(1,"$250"),(2,"$300"),(3,"$100")};
  arrows;
```

```
} : D

// this is for the delta functor
instance I = delta F K
```

Pi:

sketch from https://www.youtube.com/watch?v=Q0m8baqBrk4 (Ryan Wisnesky - A Functorial Query Language)

Pi in FQL (guess):

```
schema C = {
 nodes
     N1,
     N2;
 attributes
        name0 : N1 -> string,
        salary0 : N1 -> string,
        age0 : N2 -> int;
```

```
  arrows;
  equations;
}

schema D = {
  nodes
        N;
  attributes
         name : N -> string,
         salary : N -> string,
         age : N -> int;
  arrows;
  equations;
}

mapping F = {
 nodes
   N1 -> N,
   N2 -> N;
 attributes
   name0 -> name,
   salary0 -> salary,
   age0 -> age;
 arrows;
} : C -> D


// this is for the sigma and pi functors
instance J = {
  nodes
        N1 -> {1,2,3},
        N2 -> {1,2,3};
  attributes
        name0 -> {(1,"Bob"),(2,"Sue"),(3,"Alice")},
        salary0 -> {(1,"$250"),(2,"$300"),(3,"$100")},
        age0 -> {(1,20),(2,20),(3,30)};
  arrows;
} : C

// this is for the pi functor
instance L = pi F J

instance Q = delta F L

transform monad_counit = Q.coreturn

instance M = pi F Q
```

```
transform monad_unit = M.return
```

[5] David I. Spivak, Functorial Data Migration, https://arxiv.org/pdf/1009.1166.pdf
[6] Functorial Query Language (Archive) , https://github.com/CategoricalData/FQL, Accessed Oct. 19, 2020 (FQL is the predecessor to CQL which is here: https://github.com/CategoricalData/CQL)
[7] Ryan Wisnesky - A Functorial Query Language, Boston Haskell , https://www.youtube.com/watch?v=Q0m8baqBrk4 , Feb 15, 2015
[8]David Spivak: Categorical Databases, Topos, https://www.youtube.com/watch?v=bk36__qkhrk , 2019/02/27

Other happenings:

Josh Shinaver (Uber) and Ryan Wisnesky (Conexus)

https://eng.uber.com/dragon-schema-integration-at-uber-scale/ [Dragon: Schema Integration at Uber Scale, Uber Engineering]

https://arxiv.org/abs/1909.04881 [Algebraic Property Graphs, Joshua Shinavier, Ryan Wisnesky]

https://www.meetup.com/Category-Theory/events/lcmnvrybclbgb/ [Algebraic Property Graphs ,Tuesday, August 4th]

https://us2ts.org/2020/keynote-joshua-shinavier [3rd U.S. Semantic Technologies Symposium , March 9-11, 2020]

https://www.slideshare.net/joshsh/algebraic-property-graphs-gql-community-update-oct-9-2019

See https://github.com/CategoricalData/CQL >> APG example

for the coq (https://github.com/coq/coq) part of it see https://github.com/CategoricalData/APG

Josh Shinaver (Tinkerpop), Marko Rodriguez (former Tinkerpop) and Ryan Wisnesky (Conexus)

Tinkerpop is working around the GQL community:
https://gql.today/
https://www.w3.org/Data/events/data-ws-2019/report.html
https://www.gqlstandards.org/
https://www.gqlstandards.org/community-updates

Dr. Ryan Wisnesky is working with Dr. Marko Rodriguez (Tinkerpop Gremlin) on the mmADT project.

"mm-ADT™ is a distributed virtual machine capable of integrating a diverse collection of data processing technologies. This is made possible

via three language-agnostic interfaces: language, process, and storage." [9]

Comparing Gremlin to mmADT[9]. Gremlin started in the days of exploring a path algebra for multi-relational graphs [10] while mmADT is a bull traversing a Cayley graph.

[9] http://www.mm-adt.org/ , https://www.slideshare.net/slidarko/mmadt-a-multimodel-abstract-data-type
[10] Rodriguez, M; Peter, N; A Path Algebra for Multi-Relational Graphs, https://arxiv.org/abs/1011.0390

Dr. Ryan Wisnesky / Conexus is also working with Statebox, which is like a “blockchain” with category theory.