Erik Meijer
Microsoft SQL Server

Gavin Bierman
Microsoft Research Cambridge

Towards a mathematical model for noSQL

NoSQL Took Away The Relational Model And Gave Nothing Back

Benjamin Black
10/26/2010 Palo Alto NoSQL meetup

What he meant:

NoSQL systems are lacking a standard model for describing and querying. Developing one should be a high priority task.

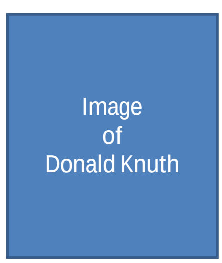
noSQL
is dual to
SQL

Objects
vs
Tables

Objects

I do consider assignment statements and pointer variables to be among computer science's most valuable treasures.

Donald Knuth

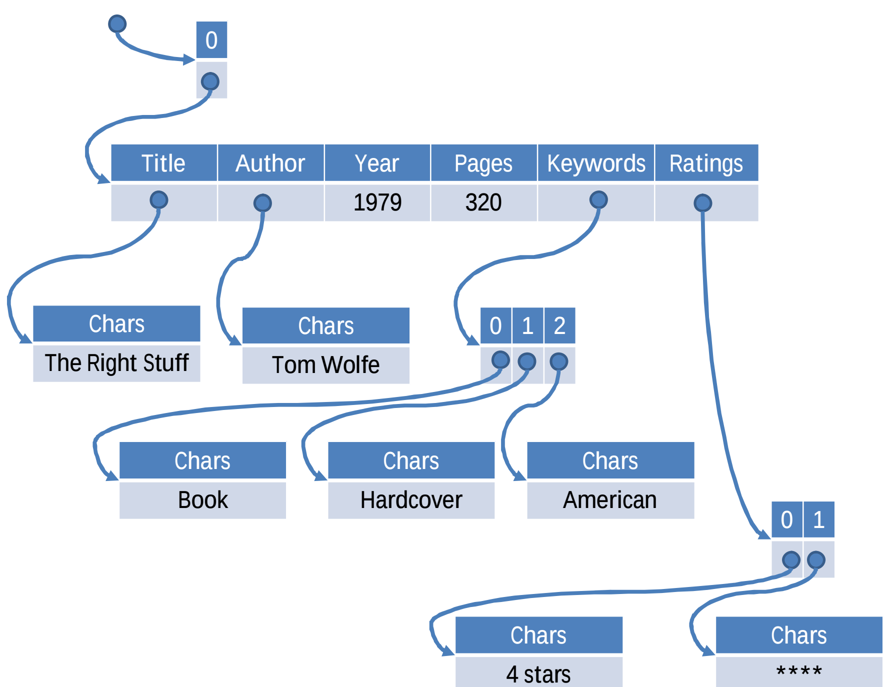


Amazon SimpleDB Sample Query Dataset

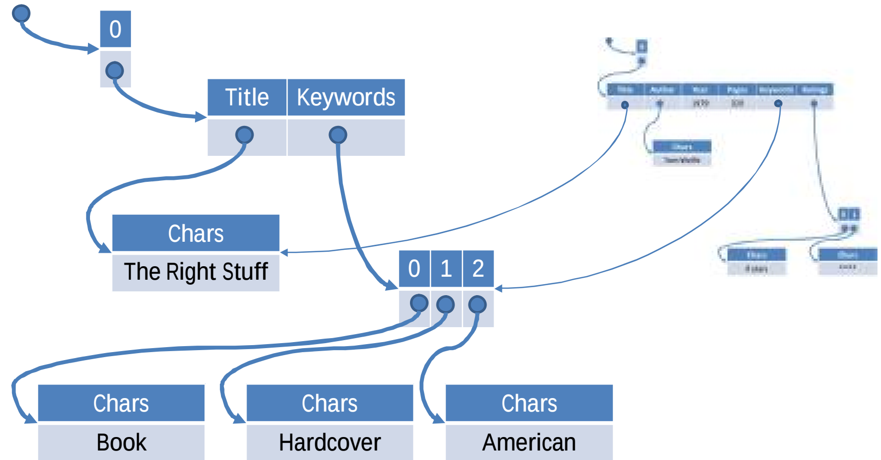
```
class Product
{
   string Title;
   string Author;
   int Year;
   int Pages;
   IEnumerable<string> Keywords;
   IEnumerable<string> Ratings;
}

var _1579124585 = new Product
{
  Title = “The Right Stuff”,
  Author = “Tom Wolfe”,
  Year = 1979,
  Pages = 304,
  Keywords = new[]{ “Book”, “Hardcover”, “American” },
  Ratings = new[]{ “****”, “4 stars” },
}

var Products = new[]{ _1579124585 };
```



```
var q = from product in Products
        where product.Ratings.Any(rating => rating == "****")
        select new{ product.Title, product.Keywords };
```



Tables

The relational model is a particularly suitable structure for the truly casual user (i.e., a non-technical person who merely wishes to interrogate the database, for example a housewife who wants to make enquiries about this week's best buys at the supermarket). In the not too distant future the majority of computer users will probably be at this level.

C.J. Date & E.F. Codd

Image of C.J. Date

JOE CELKO'S
TREES AND
HIERACHIES
IN SQL FOR
SMARTIES

Image
Of
Joe Celko

JOE CELKO'S
SQL FOR
SMARTIES
advanced SQL
Programming
Third Edition

http://troels.arvin.dk/db/rdbms/links/#hierarchical

```
table Products
{
   int ID;
   string Title;
   string Author;
   int Year;
   int Pages;
}

table Keywords
{
   int ID;
   string Keyword;
   int ProductID;
}

table Ratings
{
   int ID;
   string Rating;
   int ProductID;
}
```

```
Products.Insert
         ( 1579124585
         , "Tom Wolfe"
         , 1979
         , 304
         );

Keywords.Insert
         ( 4711
         , "Book"
         , 1579124585
         );
Keywords.Insert
         ( 1843
         , "Hardcover"
         , 1579124585
         );
Keywords.Insert
         ( 2012
         , "American"
         , 1579124585
         );
```

```
Ratings.Insert
         ( 787
         , "****"
         , 1579124585
         );
Ratings.Insert
         ( 747
         , "4 stars"
         , 1579124585
         );
```

In SQL rows
are not expressible

Ratings

ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

Products

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

Keywords

ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585

Referential Integrity

Maintained by the environment

ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

Foreign key must have corresponding primary key

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

Primary key must be unique

```
var q = from product in Products
        from rating in Ratings
        where product.ID == rating.ProductId
              && rating == "****"
        from keyword in Keywords
        where product.ID == keyword.ProductID
        select new{ product.Title, keyword.Keyword };
```

Title	Keyword
The Right Stuff	Book
The Right Stuff	Hardcover
The Right Stuff	American

```
var q = from product in Products
        join rating in Ratings
        on product.ID equals rating.ProductId
        where rating == "****"
        select product into FourStarProducts
        from fourstarproduct in FourStarProducts
        join keyword in Keywords
        on product.ID equals keyword.ProductID
        select new{ product.Title, keyword.Keyword };
```

In mathematics, semantics, and philosophy of language, the Principle of Compositionality is the principle that the meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them.

Gottlob Frege 1848-1925

Objects

Fully compositional

```
value ::= scalar
          new { ... , name = value, ... }
```

Tables

Non compositional

```
value ::= new { ... , name = scalar, ... }
```

Tables

Non compositional

Query results denormalized
Query can only return single table
No recursion (but have CTEs)

NULL semantics a mess

Sum(1,NULL) = 1
1+NULL = NULL

Impedance Mismatch

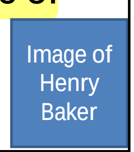
The problem with having two languages is "impedance mismatch " One mismatch is conceptual -the data language and the programming languages might support widely different programming paradigms. [...] The other mismatch is structural -the languages don't support the same data types, [...]

George Copeland & David Maier 1984

The "relational" data model, enunciated by Ted Codd in a landmark 1970 article, was a major advance over DBTG. The relational model unified data and metadata so that there was only one form of data representation. It defined a non-procedural data access language based on algebra or logic. It was easier for end-users to visualize and understand than the pointers-and-records-based DBTG model. Programs could be written in terms of the "abstract model" of the data, rather than the actual database design; thus, programs were insensitive to changes in the database design.

Jim Gray

Codd's relational theory dressed up these concepts with the trappings of mathematics (wow, we lowly Cobol programmers are now *mathematicians!*) by calling files *relations,* records *rows,* fields *domains,* and merges *joins.*
Computing history will consider the past 20 years as a kind of Dark Ages of commercial data processing in which the religious zealots of the Church of Relationalism managed to hold back progress until a Renaissance rediscovered the Greece and Rome of pointer-based databases. Database research has produced a number of good results, but the relational database is not one of them.



Henry G. Baker

LINQ to SQL MSDN documentation

LINQ to SQL provides a runtime infrastructure for managing relational data as objects without losing the ability to query. Your application is free to manipulate the objects while LINQ to SQL stays in the background tracking your changes automatically.

Entity Framework MSDN documentation

When one takes a look at the amount of code that the average application developer must write to address the impedance mismatch across various data representations (for example objects and relational stores) it is clear that there is an opportunity for improvement.

```
[Table(name="Products")]
class Product
{
   [Column(PrimaryKey=true)]int ID;
   [Column]string Title;
   [Column]string Author;
   [Column]int Year;
   [Column]int Pages;

   private EntitySet<Rating> _Ratings;
   [Association( Storage="_Ratings"
   , ThisKey="ID", OtherKey="ProductID"
   , DeleteRule="ONDELETECASCADE")]
   ICollection<Rating> Ratings{ ... }

   private EntitySet<Keyword> _Keywords;
   [Association( Storage="_Keywords",
   , ThisKey="ID", OtherKey="ProductID",
   , DeleteRule="ONDELETECASCADE")]
   ICollection<Keyword> Keywords{ ... }
}
```

```
[Table(name="Keywords")]
class Keyword
{
   [Column(PrimaryKey=true)]int ID;
   [Column]string Keyword;
   [Column(IsForeignKey=true)]int ProductID;
}

[Table(name="Ratings")]
class Rating
{
   [Column(PrimaryKey=true)]int ID;
   [Column]string Rating;
   [Column(IsForeignKey=true)]int ProductID;
}
```

And we did not even talk about inheritance yet.

```
var q = from product in Products
        from rating in Ratings
        where product.ID == rating.ProductId
              && rating == "****"
        from keyword in Keywords
        where product.ID == keyword.ProductID
         select new{ product.Title, keyword.Keyword };
```

```
var q = from product in Products
        where product.Ratings.Any(rating =>  rating.Rating == "****")
        select new{ product.Title, product.Keywords };
```

ID	Title
1579124585	The Right Stuff

ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585

Indexes Recover Nesting

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

ID	from rating in Ratings where ID = rating.ID select rating.ID		from keyword in Keywords where ID = keyword.ID select keyword.ID		
1579124585	787	747	4711	1843	2012

ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585

ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

Keywords		
4711	1843	2012

Ratings	
787	747

ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585

ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

Normalization is for Sissies

Pat Helland

Image of Pat Helland

Ad-hoc queries

```
from p1 in Products
from p2 in Products
where p1.Title.Length == p2.Author.Length
select new{ p1, p2 };
```

Does not really work:
- $O(n^2)$
- No referential integrity

Ad-hoc queries don’t scale

```
from p1 in WWW
from p2 in WWW
where p2.Contains(p1.URL)
select new{ p1, p2 };
```

Sorting the whole Web
Might be a bit of a challenge

Designer

Remove original hierarchical structure into normalized data

App Developer

Recover original hierarchical structure from normalized data

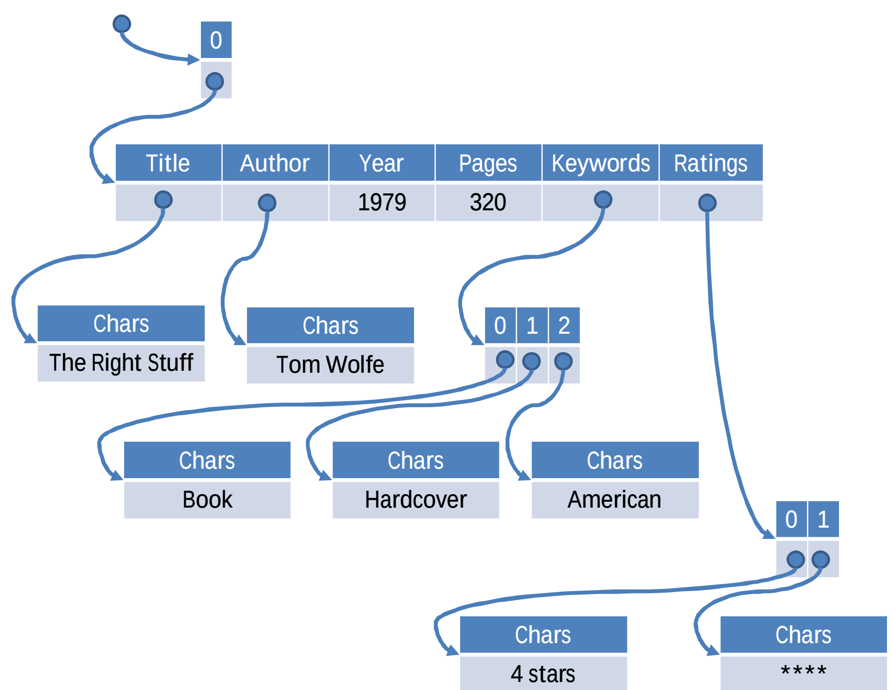
Database Implementer

Recover original hierarchical structure from normalized data

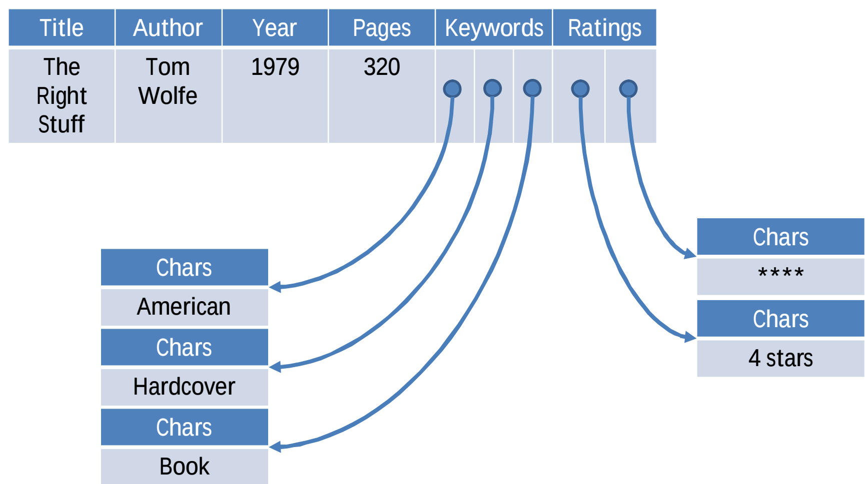
PEACE

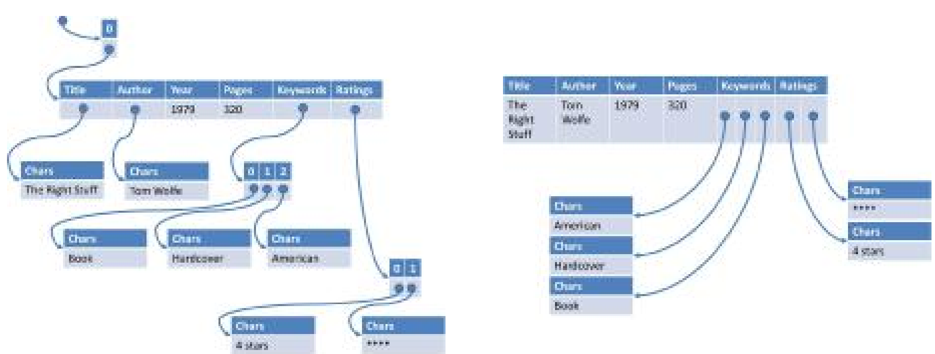
not WAR

http://en.wikipedia.org/wiki/Math_Rescue



ignore identity of collections





ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

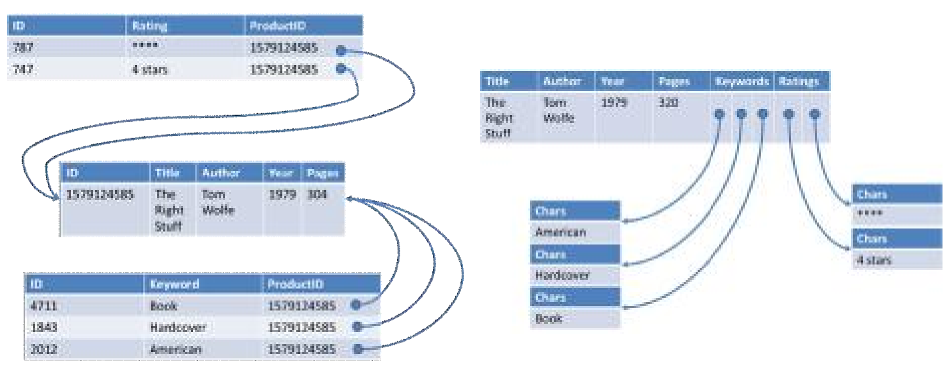
ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585

Draw relationships as arrows

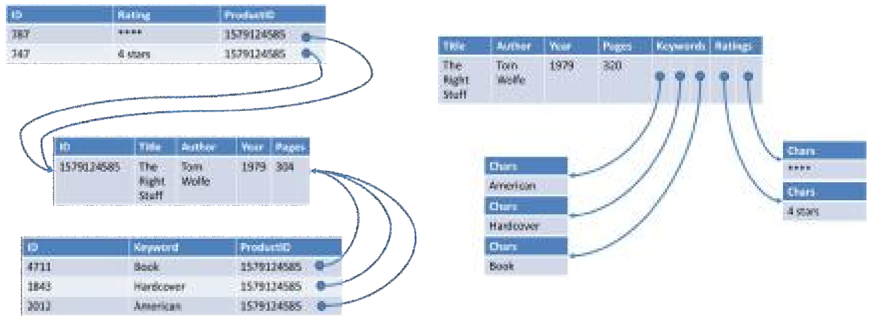
ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

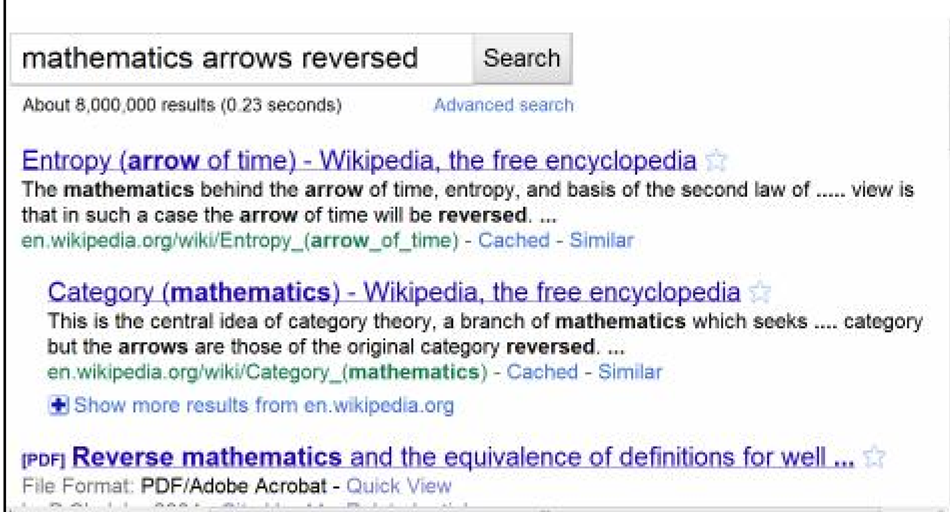
ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585



Spot the differences



- Arrows are reversed
- Identity extensional/intensional



Categories, objects, and morphisms

Main articles: Category (mathematics) and Morphism

A *category* C consists of the following three mathematical entities:

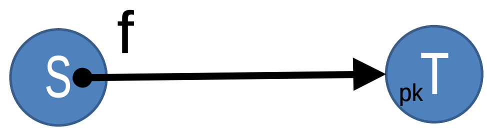
- A class ob(C), whose elements are called *objects*;
- A class hom(C), whose elements are called morphisms or maps or *arrows*. Each morphism f has a unique *source object* a and *target object* b. We write $f: a \to b$, and we say "f is a morphism from a to b". We write hom(a, b) (or Hom(a, b), or $\mathrm{hom}_C(a, b)$, or Mor(a, b), or $C(a, b)$) to denote the *hom-class* of all morphisms from a to b.
- A binary operation ∘, called *composition of morphisms*, such that for any three objects a, b, and c, we have hom(a, b) × hom(b, c) → hom(a, c). The composition of $f: a \to b$ and $g: b \to c$ is written as $g \circ f$ or gf [2], governed by two axioms:
  - Associativity: If $f: a \to b$, $g: b \to c$ and $h: c \to d$ then $h \circ (g \circ f) = (h \circ g) \circ f$, and
  - Identity: For every object x, there exists a morphism $1_x: x \to x$ called the *identity morphism for x*, such that for every morphism $f: a \to b$, we have $1_b \circ f = f = f \circ 1_a$.

From these axioms, it can be proved that there is exactly one identity morphism for every object. Some authors deviate from the definition just given by identifying each object with its identity morphism.

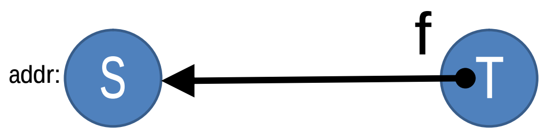
Relations among morphisms (such as $fg = h$) are often depicted using commutative diagrams, with "points" (corners) representing objects and "arrows" representing morphisms.

The definitions of categories and functors provide only the very basics of categorical algebra; additional important topics are listed below. Although there are strong interrelations between all of these topics, the given order can be considered as a guideline for further reading.

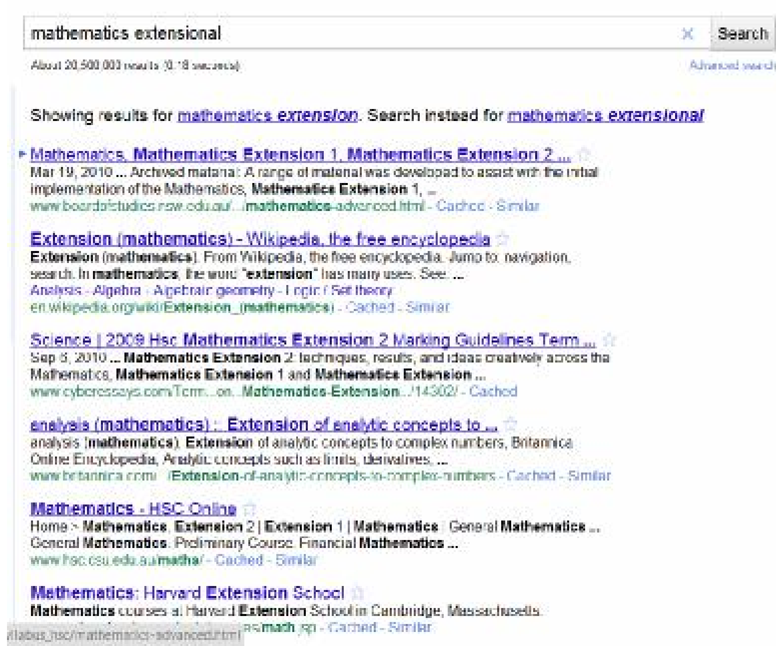
- The functor category D^C has as objects the functors from C to D and as morphisms the natural transformations of such functors. The Yoneda lemma is one of the most famous basic results of category theory; it describes representable functors in functor categories.
- Duality: Every statement, theorem, or definition in category theory has a *dual* which is essentially obtained by "reversing all the arrows". If one statement is true in a category C then its dual will be true in the dual category C^{op}. This duality, which is transparent at the level of category theory, is often obscured in applications and can lead to surprising relationships.
- Adjoint functors: A functor can be left (or right) adjoint to another functor that maps in the opposite direction. Such a pair of adjoint functors typically arises from a construction defined by a universal property; this can be seen as a more abstract and powerful view on universal properties.



ForeignKey(f,s) = PrimaryKey(t)

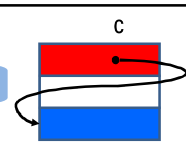


Address(s) = Property(f,t)



In logic and mathematics, an *intensional* definition gives the meaning of a term by specifying all the properties required to come to that definition, that is, the necessary and sufficient conditions for belonging to the set being defined.

An *extensional* definition of a concept or term formulates its meaning by specifying its extension, that is, every object that falls under the definition of the concept or term in question.



Objects

subject verb direct object

A memory location contains an *object*

A pointer is the memory location of some object

Memory location is not part of the object

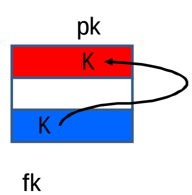
subject verb direct object

Rows

A *row* has a primary key

A foreign key is the value of a primary key

Primary key is part of a row



F-algebra

From Wikipedia, the free encyclopedia

In mathematics, specifically in category theory, an F**-algebra** for an endofunctor

$$F : \mathcal{C} \longrightarrow \mathcal{C}$$

is an object A of $\mathcal{C}$ together with a $\mathcal{C}$-morphism

$$\alpha : FA \longrightarrow A .$$

In this sense F-algebras are dual to F-coalgebras.

F-coalgebra

From Wikipedia, the free encyclopedia

In mathematics, specifically in category theory, an F**-coalgebra** for an endofunctor

$$F : \mathcal{C} \longrightarrow \mathcal{C}$$

is an object A of $\mathcal{C}$ together with a $\mathcal{C}$-morphism

$$\alpha : A \longrightarrow FA .$$

In this sense F-coalgebras are dual to F-algebras.

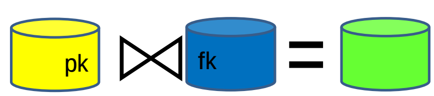
Relational Algebra

Algebraic: Table⋈Table à Table

Join *constructs* new row by combining other rows

A Relational Model of Data for Large Shared Data Banks

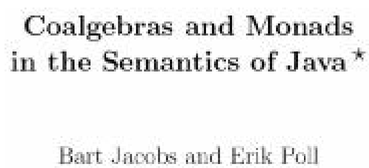
E. F. Codd
IBM Research Laboratory, San Jose, California



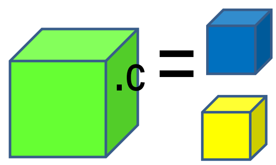
Object CoAlgebra

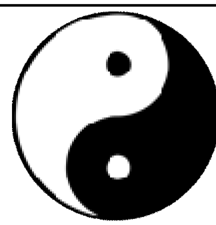
coAlgebraic: Object•Member **à** Object*

Member access *destructs* existing object into constituent objects


Coalgebras and Monads
in the Semantics of Java*

Bart Jacobs and Erik Poll





Key-Value Store Is Dual To Primary/Foreign-key Store

noSQL
is
coSQL

noSQL and SQL are not in conflict,
like good and evil.

They are two opposites
that co-exist in harmony
and can transmute into each other.

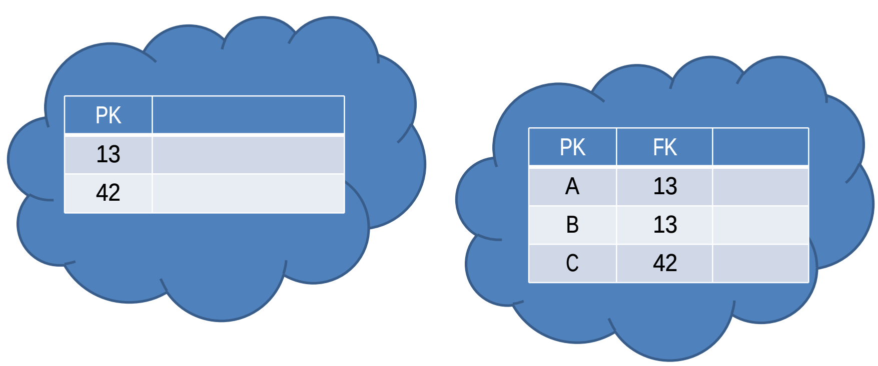
Like yin (open **è** noSQL)
and yang (closed **è** SQL).

Consequences of Duality

If a statement T is true in C

Then its dual co(T) is true in co(C)

SQL	coSQL
Children point to parents	Parents point to children
Closed world	Open world
Entities have identity (extensional)	Environment determines identity (intensional)
Synchronous (ACID)	Asynchronous (BASE)
Environment coordinates changes (transactions)	Entities responsible to react to changes (eventually consistent)
Not compositional	Compositional
Query optimizer	Developer/pattern

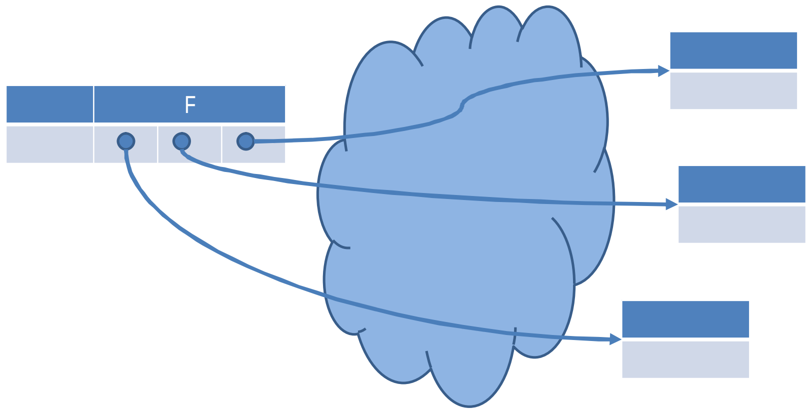


Open world

Cannot join, build indexes

Cannot coordinate transactions

Cannot maintain referential integrity



Pre-computed indexes
Eventually consistent
Weak pointers (expect 404)

Life beyond Distributed Transactions: an Apostate's Opinion

Entities are collections of named (keyed) data which may be atomically updated within the entity but never atomically updated across entities.

Pat Helland

SimpleDB Datamodel

Domain ::= {Item; Row}*
Row ::= { …; Attribute = Value+; … }
Value ::= string | key

Title	Author	Year	Pages	Keywords	Ratings
The Right Stuff	Tom Wolfe	1979	320	Hardcover	****
				American	4 stars
				Book	

Actual mathematical dual of
flat relational tables with scalars in columns

SimpleDB Downside

Title	Author	Year	Pages	Keywords	Ratings
The Right Stuff	Tom Wolfe	1979	320	Hardcover	****
				American	4 stars
				Book	

No way to retrieve multi-valued attributes using select query. Needs two round trips (can batch writes).

```
sdb.GetAttributes(new GetAttributesRequest
{
   AttributeName = {"Keyword", "Rating"},
   DomainName="Books",
   ItemName = "…itemName() from query …",
});
```

HTML 5 Storage

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key,
                              in any data);
  deleter void removeItem(in DOMString key);
  void clear();
}
```

Actual mathematical dual of relational tables with blobs

What About SQL (the query language)

More Category Theory

Monads as Kleisli triples

Rather than focusing on a specific T, we want to find the general properties common to all notions of computation, therefore we impose as only requirement that *programs* should form a category. The aim of this section is to convince the reader, with a sequence of informal argumentations, that such a requirement amounts to say that T is part of a Kleisli triple $(T, \eta, \_^*)$ and that the category of programs is the Kleisli category for such a triple.

Definition 1.2 ([**Man76**]) *A* **Kleisli triple** *over a category $\mathcal{C}$ is a triple $(T, \eta, \_^*)$, where $T: \mathrm{Obj}(\mathcal{C}) \to \mathrm{Obj}(\mathcal{C})$, $\eta_A: A \to TA$ for $A \in \mathrm{Obj}(\mathcal{C})$, $f^*: TA \to TB$ for $f: A \to TB$ and the following equations hold:*

- $\eta_A^* = \mathrm{id}_{TA}$
- $\eta_A; f^* = f$ *for* $f: A \to TB$
- $f^*; g^* = (f; g^*)^*$ *for* $f: A \to TB$ *and* $g: B \to TC$.

A Kleisli triple satisfies the **mono requirement** *provided η_A is mono for $A \in \mathcal{C}$.*

Intuitively η_A is the *inclusion* of values into computations (in several cases η_A is indeed a mono) and f^* is the *extension* of a function f from values to computations to a function from computations to computations, which first evaluates a computation and then applies f to the resulting value. In

Query Processor

select F(a,b)
from as as a
from bs as b
where P(a,b)

Turns pretty Syntax

$\pi_F (\sigma_P (asXbs))$

Into scary math

What is the *interface* that the relational algebra implements?

We want to query both SQL and noSQL using the same query language

And every other data source as well.

Picture of Ted Codd

Picture of Saunders Mac Lane

Sets **à** “Collections”

Tuples **à** “Generics”

∅ :: M<T>
∪ :: M<T>xM<T> **à** M<T>
{_} :: T **à** M<T>

σ_P :: M<T>x(T **à** bool) **à** M<T>
π_F :: M<T>x(T **à** S) **à** M<S>
X :: M<T>xM<S> **à** M<TxS>

Correlated Subqueries

SelectMany ::
M<T>x(T **à** M<S>) **à** M<S>

σ_P(as) =
as.SelectMany(λa **à**
P(a)?{a}: ∅)

Correlated Subqueries

π_F(as) =
as.SelectMany(λa **à** {F(a)})

as X bs =
as.SelectMany(λa **à**
$\sigma_{\lambda b\ \text{à}\ (a,b)}$(bs))

One important twist

SelectMany ::
M<T>
X
(Expr<T **à** M<S>>)
à
M<S>

Intensional representation of code

Recognize the Monads?

M<_> **à** Functor
SelectMany **à** bind
{_} **à** return/η

μ :: M<M<T>> **à** M<T>
μ tss = tss.SelectMany(λts **à** ts)

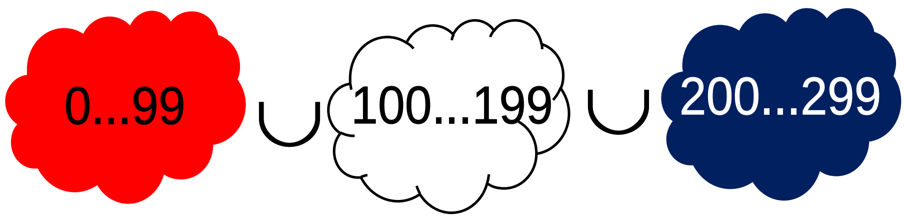
LINQ == Monads

Syntactic sugar for monad comprehensions

Data source “implements” monadic interface (pattern)

One query syntax over multiple data models

coSQL naturally allows extreme horizontal partitioning



Bird’s First Homomorphism Lemma
1987

Picture of Richard Bird

A function
  h :: M<A> **à** B
is a homomorphism wrt to ∪ iff
  h = (⊕/) • (f*) -- “/” is reduce, “*” is map
for some
  f :: A **à** B
and
  ⊕ :: BxB **à** B

For the rest of us
Every LINQ query can be executed as a MapReduce computation

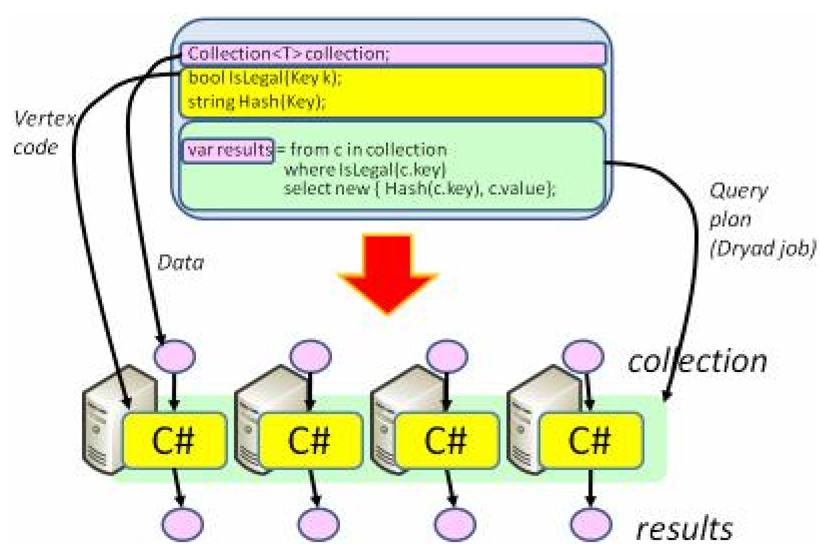
Google's MapReduce Programming Model -- Revisited

Picture of
Ralf Lämmel

```
class MapReduce<k1, k2, v1, v2, v3>
{
   IEnumerable<KeyValuePair<k2, v2>> Map(k1 Key, v1 Value);
   v3 Reduce(k2 Key, IEnumerable<v2> Values);

   IEnumerable<KeyValuePair<k2, v3>> MapReduce
  ( IEnumerable<KeyValuePair<k1, v1>> Input)
  {...}
}
```

DryadLINQ



We Are Hiring

Databases

Business

LINQ

Hacker

Category Theory

Functional Programming

Distributed Systems