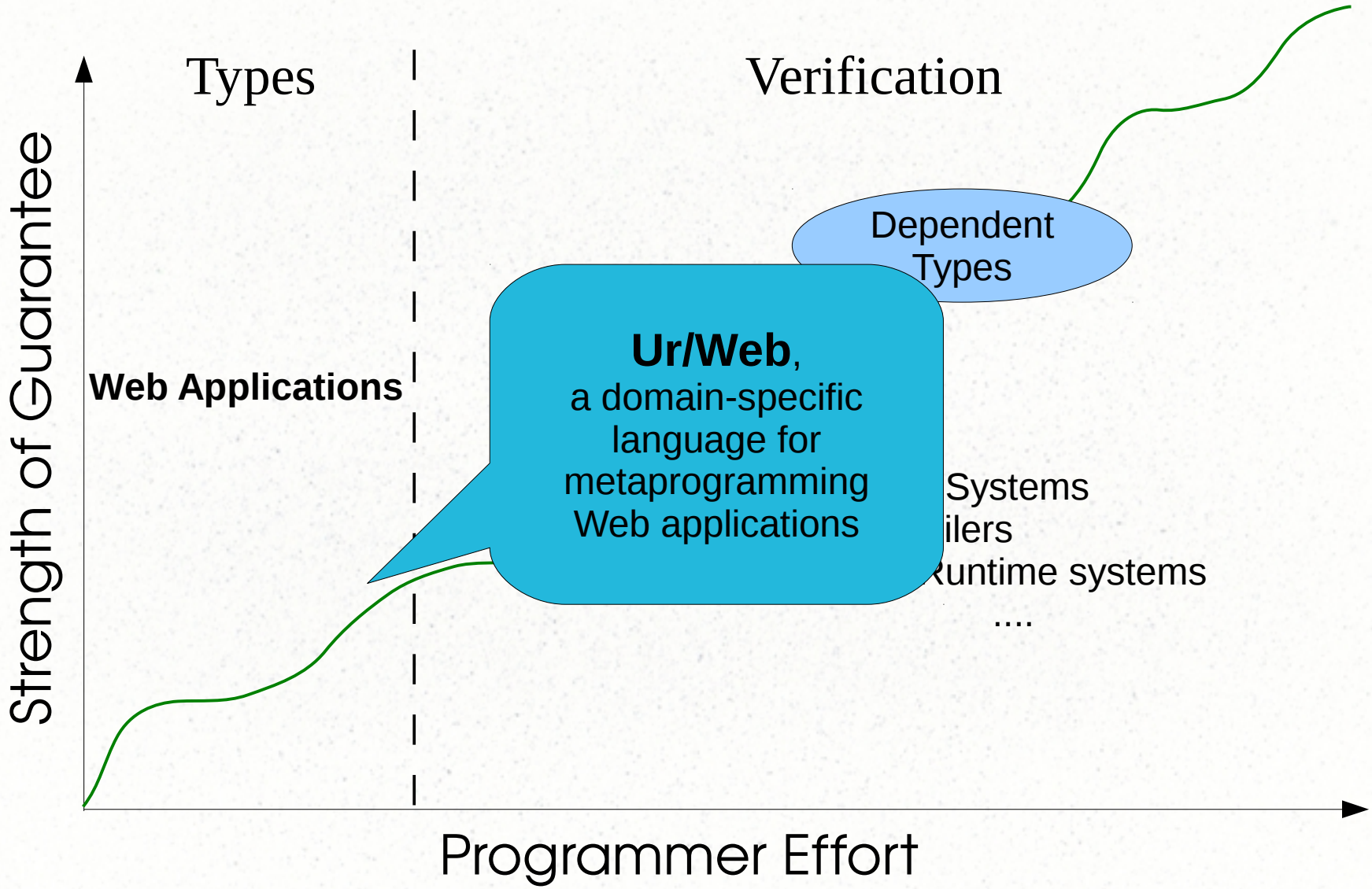
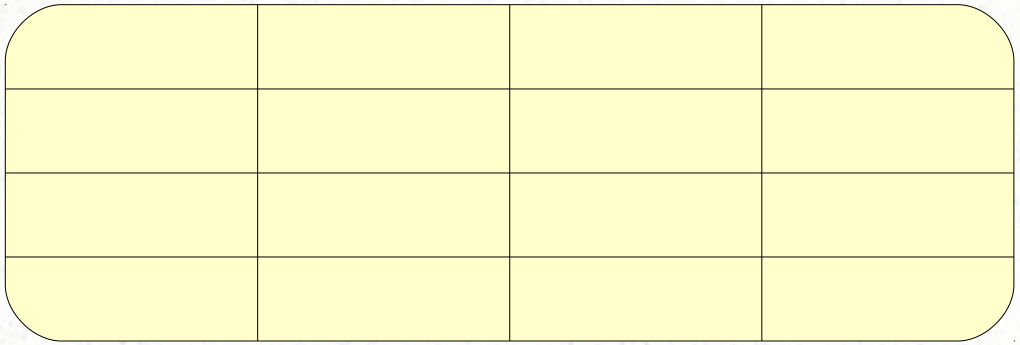
Ur: Statically-Typed Metaprogramming with Type-Level Record Computation

Adam Chlipala
PLDI 2010


Types
Verification
Strength of Guarantee
Web Applications
Dependent Types
Ur/Web, a domain-specific language for metaprogramming Web applications
Systems
ilers
untime systems
....
Programmer Effort

Object-Relational Mapping

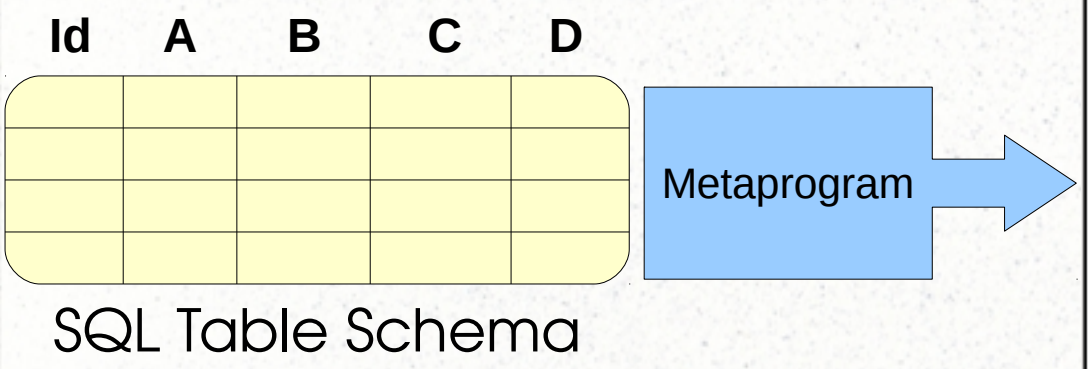


SQL Table Schema

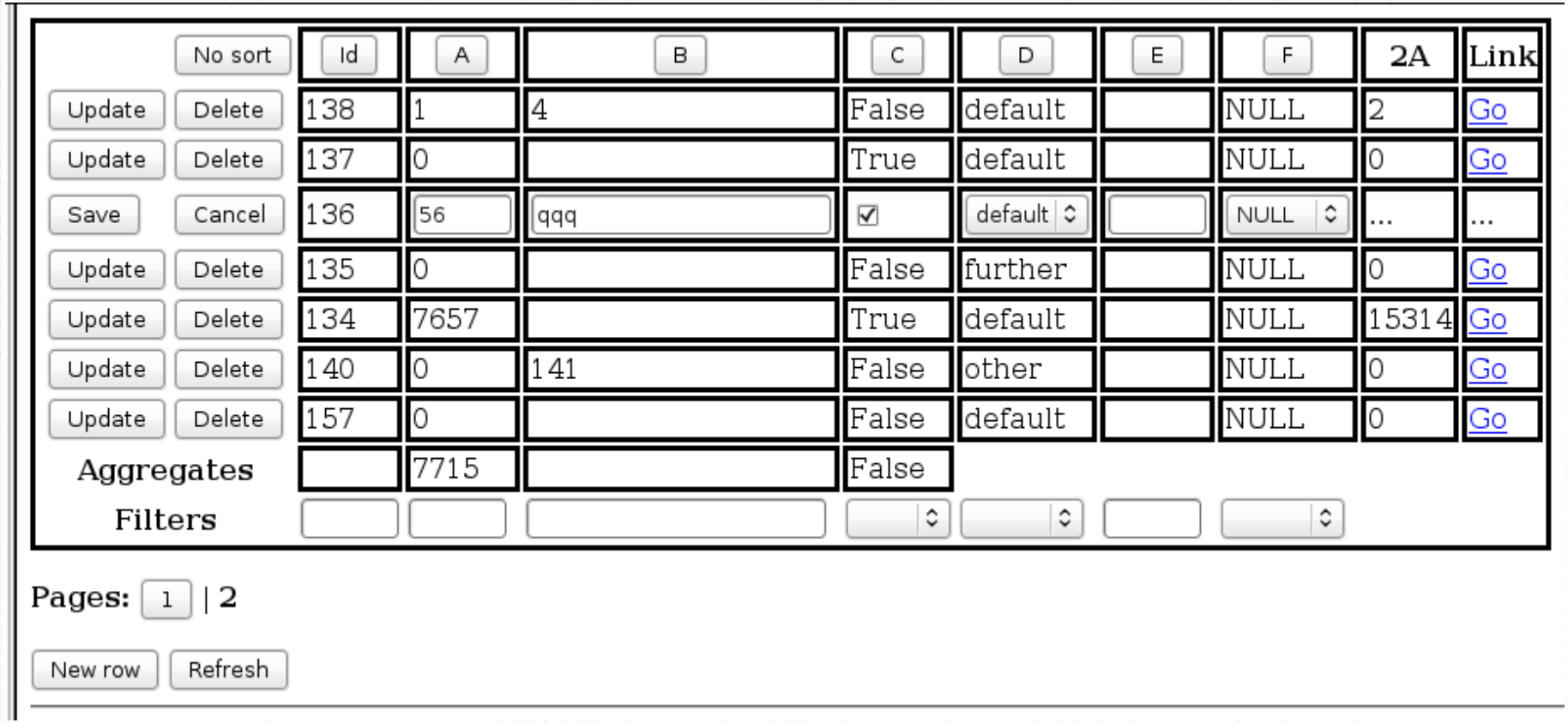
Object-Oriented Interface

- Add row
- Modify row
- Search
- ...

Automatic Admin Interface



In-Browser Spreadsheet



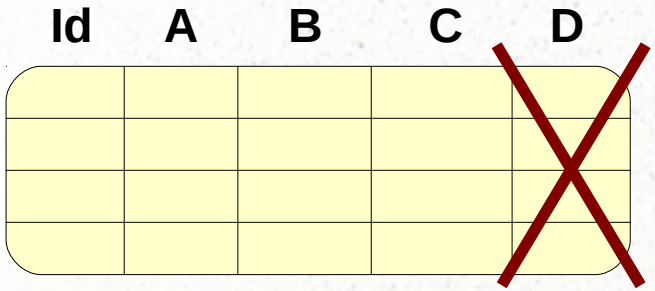
	No sort	Id	A	B	C	D	E	F	2A	Link
Update	Delete	138	1	4	False	default		NULL	2	Go
Update	Delete	137	0		True	default		NULL	0	Go
Save	Cancel	136	56	qqq	☑	default		NULL	...	...
Update	Delete	135	0		False	further		NULL	0	Go
Update	Delete	134	7657		True	default		NULL	15314	Go
Update	Delete	140	0	141	False	other		NULL	0	Go
Update	Delete	157	0		False	default		NULL	0	Go
Aggregates			7715		False					
Filters										

Pages: 1 | 2

New row | Refresh

Ad-Hoc Code Generation?

Edit some source files to customize....
Now change the database schema....



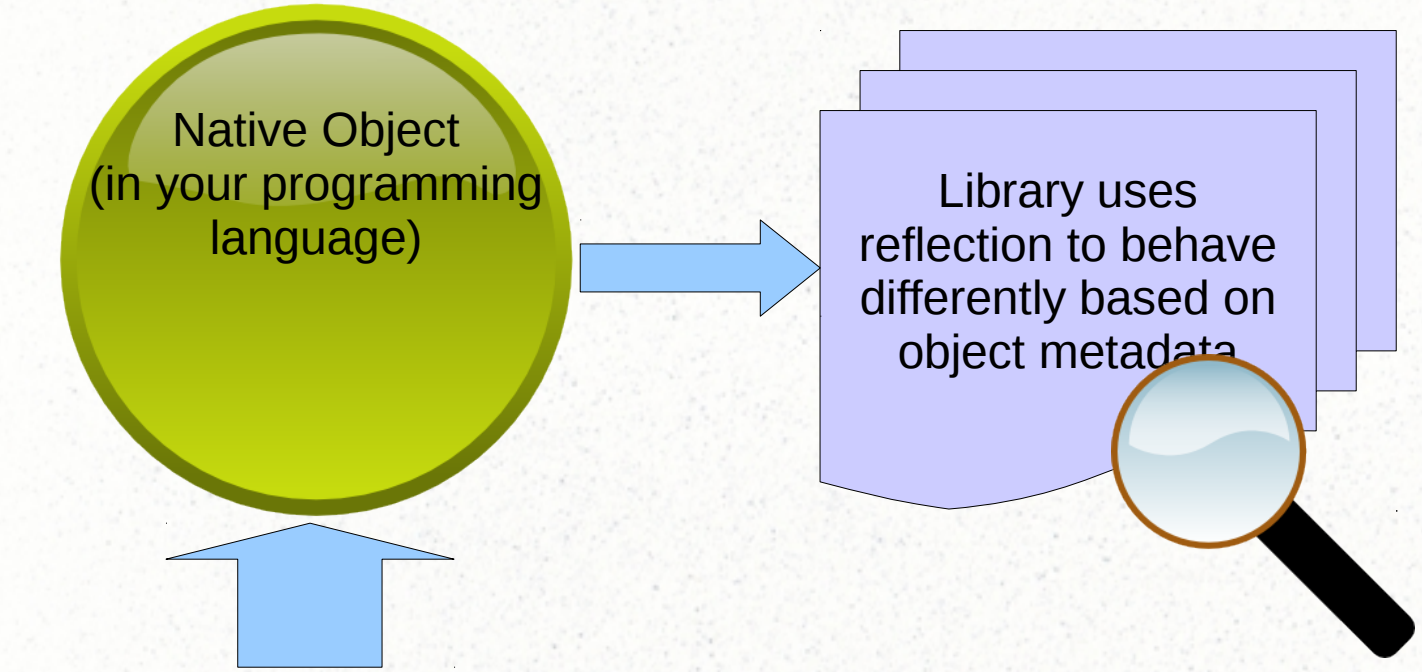
SQL Table Schema

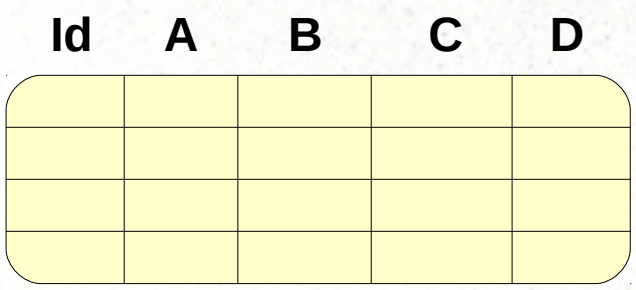
Metaprogram

```
app
  models
    post.rb
  views
    posts
      index.html.erb
      show.html.erb
      ....
  config
    routes.rb
  ....
```

```
script/generate scaffold T
  Id:int A:string B:float
      C:string D:int
```

Run-Time Reflection?



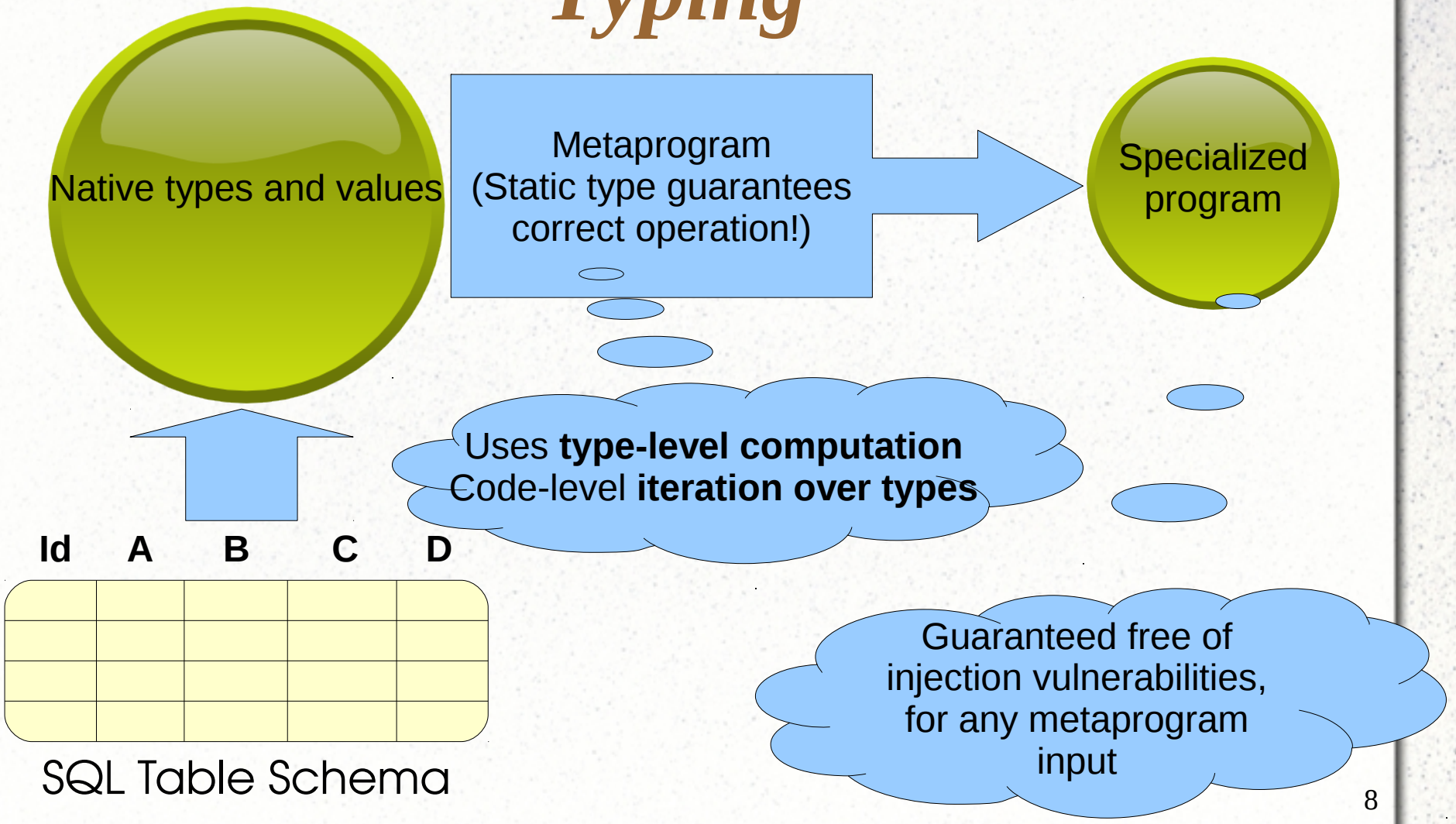


SQL Table Schema

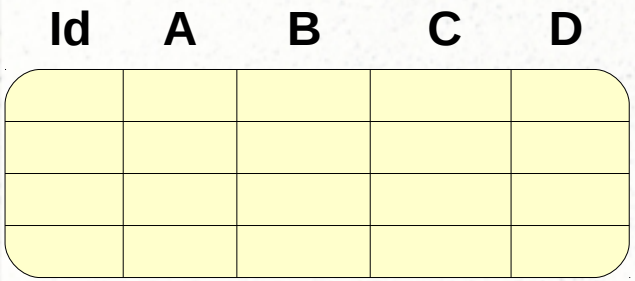
How do we know that this library is correct?

How can we be sure that the library never produces code vulnerable to **code injection attacks** or other **failures of abstraction**?

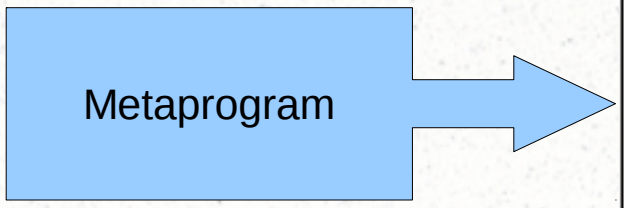
A Solution Inspired by Dependent Typing

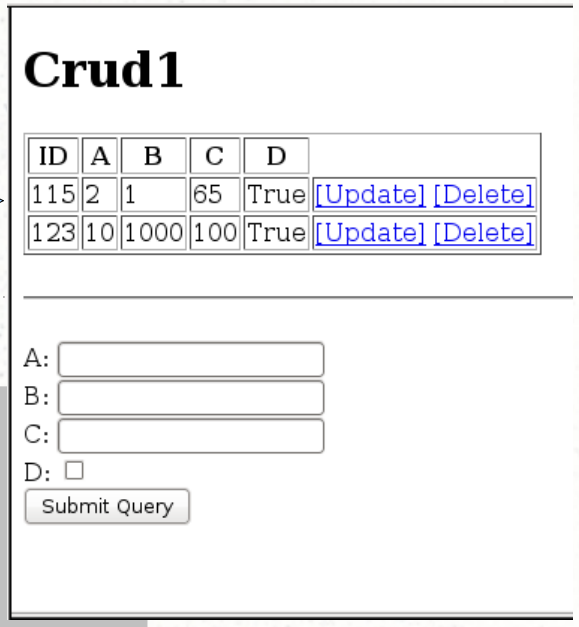


An Example Application



SQL Table Schema






```
table t1 : {Id : int, A : int, B : string,
            C : float, D : bool}
  PRIMARY KEY Id

open Crud.Make(struct
                   val tab = t1

                   val title = "Crud1"

                   val cols = {A = Crud.int "A",
                               B = Crud.string "B",
                               C = Crud.float "C",
                               D = Crud.bool "D"}
               end)
```

Iterate over the structure of this record.

What's in the Type System?

- Type-level functions: $(\lambda t.\ t \to t)$
- Type-level records: $[A = (\lambda t.\ t \to t), B = (\lambda t.\ t)]$
- Type-level record concatenation: $A +\!+ B$
- Type-level map: $\text{map}\ (\lambda t.\ t \to t)\ [A = \text{int}, B = \text{float}]$
- Disjointness constraints: $[A = \text{int}, B = \text{float}] \sim r$

- Automatic application of algebraic laws
  - E.g., $\text{map}\ f\ (\text{map}\ g\ r) = \text{map}\ (f \circ g)\ r$

Higher-Order Polymorphism

```
HtmlPage adminInterface<T>(Metadata<T> m) {
   /* iterate over m */
}
```

```
fun 'T adminInterface (m : 'T Metadata) : HtmlPage =
   (* iterate over m? *)
```

```
fun ['T :: Type] adminInterface (m : 'T Metadata)
   : HtmlPage =
   (* iterate over m? *)
```

```
fun ['T :: {Type}] adminInterface
   (m : $(map Metadata 'T))
   (it : 'T Iterator) : HtmlPage =
   it (* appropriate arguments here... *)
```

Ur/Web Available At:

`http://www.impredicative.com/ur/`

Including online demos with syntax-highlighted source code