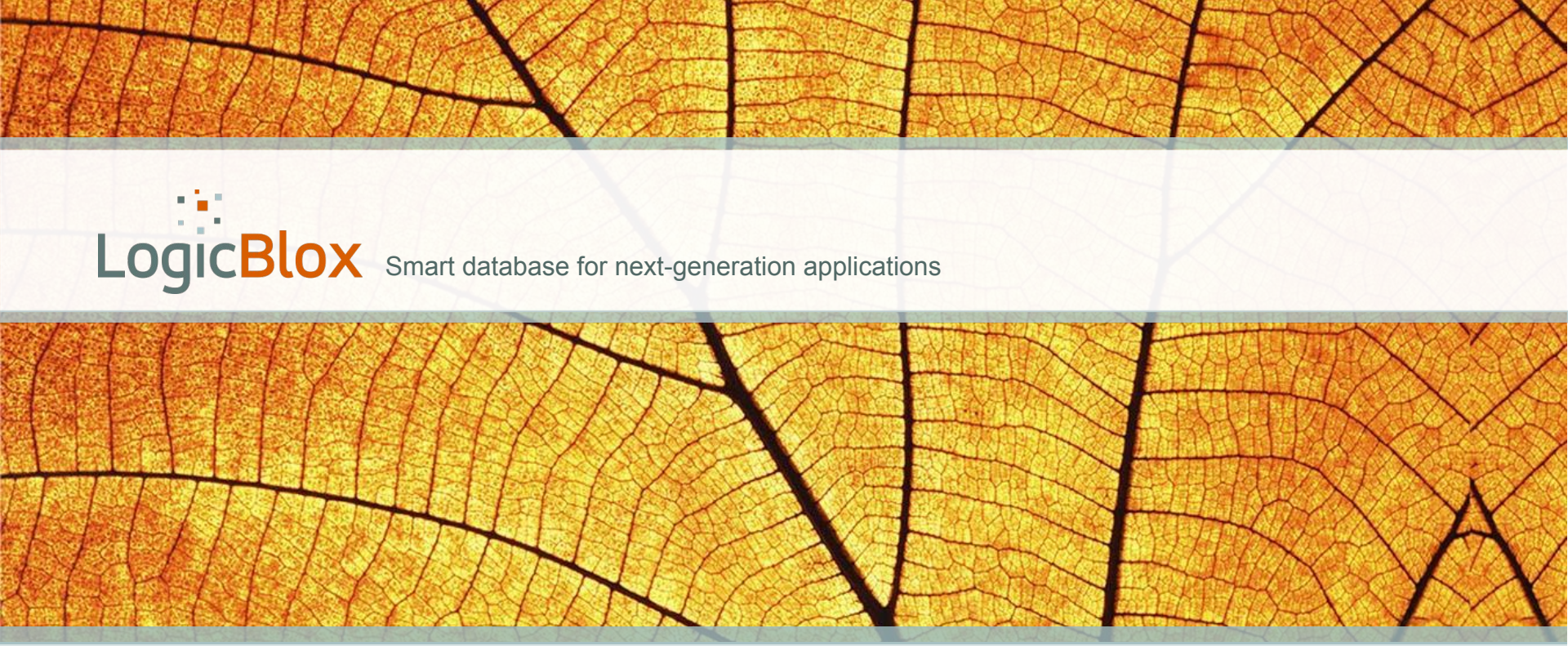


LogicBlox

Smart database for next-generation applications

LESSON 3 : ENTITIES

LOGICBLOX TRAINING

TYPES HELP CATCH BUGS

```
greetings(language, content)
  -> string(language), string(content).

person_speaks(name, language)
  -> string(name), string(language).

person_hasGreeting(name, content)
  <- person_speaks(name, 42),
     greetings(language, content).
```

TYPES HELP CATCH BUGS

```
greetings(language, content)
  -> string(language), string(content).

person_speaks(name, language)
  -> string(name), string(language).

person_hasGreeting(name, content)
  <- person_speaks(name, 42),
     greetings(language, content).
```

error: an expression of type 'int'
is used where type 'string' is needed
(code: TYPE_ERROR)

SUBTLE BUGS

```
greetings(language, content)
  -> string(language), string(content).

person_speaks(name, language)
  -> string(name), string(language).

person_hasGreeting(name, content)
  <- person_speaks(language, name),
     greetings(language, content).
```

SUBTLE BUGS

```
greetings(language, content)
  -> string(language), string(content).

person_speaks(name, language)
  -> string(name), string(language).

person_hasGreeting(name, content)
  <- person_speaks(language, name),
     greetings(language, content).
```

SUBTLE BUGS

```
greetings(language, content)
  -> string(language), string(content).

person_speaks(name, language)
  -> string(name), string(language).

person_hasGreeting(name, content)
  <- person_speaks(language, name),
     greetings(language, content).
```

ENTITIES = USER-DEFINED TYPES

- entity declaration

```
person(p) -> .
```

ENTITIES = USER-DEFINED TYPES

- entity declaration

```
person(p) -> .
```

entity predicate

ENTITIES = USER-DEFINED TYPES

- entity declaration

```
person(p) -> .
```

- using entities as type

```
person_speaks(person, language)
  -> person(person), string(language).

person_hasGreeting(person, content)
  -> person(person), string(content).
```

ENTITIES = USER-DEFINED TYPES

- entity declaration

```
person(p) -> .
```

- using entities as type

```
person_speaks(person, language)
  -> person(person), string(language).

person_hasGreeting(person, content)
  -> person(person), string(content).

person_hasGreeting(person, content)
  <- person_speaks(language, person),
     greetings(language, content).
```

ENTITIES = USER-DEFINED TYPES

- entity declaration

```
person(p) -> .
```

- using entities as type

```
person_speaks(person, language)
  -> person(person), string(language).

person_hasGreeting(person, content)
  -> person(person), string(content).

person_hasGreeting(person, content)
  <- person_speaks(language, person),
     greetings(language, content).
```

error: 'language' cannot be typed:
it cannot be a string and person
(code: TYPE_INCONSISTENT)

ENTITY REFERENCE

- entity / reference mode declaration

```
person(p), person_name(p : name) -> string(name).
```

ENTITY REFERENCE

- entity / reference mode declaration

```
person(p), person_name(p : name) -> string(name).
```

entity predicate

ENTITY REFERENCE

- entity / reference mode declaration

```
person(p), person_name(p : name) -> string(name).
```

entity predicate

reference mode (refmode) predicate

ENTITY REFERENCE

- entity / reference mode declaration

```
person(p), person_name(p : name) -> string(name).
```

entity predicate

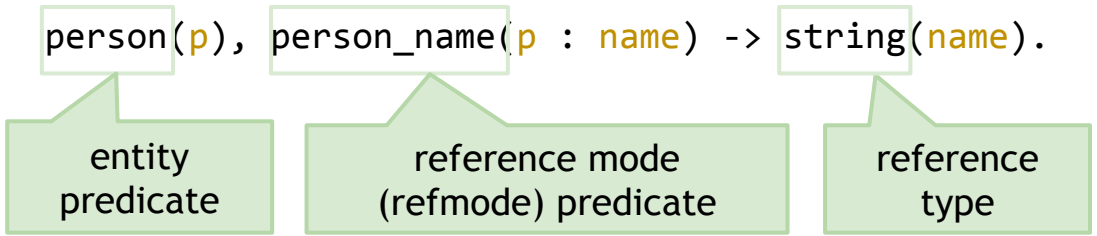
reference mode (refmode) predicate

  - refmode predicate is a one-to-one function

ENTITY REFERENCE

- entity / reference mode declaration

```
person(p), person_name(p : name) -> string(name).
```



  - refmode predicate is a one-to-one function
  - refmode value can be any primitive type

ENTITY CONSTRUCTION WITH REFMODE

```
% lb addblock helloworld '
    person(p), person_name(p : name) -> string(name).'

% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

ENTITY CONSTRUCTION WITH REFMODE

```
% lb addblock helloworld '
    person(p), person_name(p : name) -> string(name).'

% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'

% lb print helloworld person
[3] "Sarah"
```

ENTITY CONSTRUCTION WITH REFMODE

```
% lb addblock helloworld '
    person(p), person_name(p : name) -> string(name).'
```

```
% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

```
% lb print helloworld person
[3] "Sarah"
```

```
% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

ENTITY CONSTRUCTION WITH REFMODE

```
% lb addblock helloworld '
    person(p), person_name(p : name) -> string(name).'
```

```
% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

```
% lb print helloworld person
[3] "Sarah"
```

```
% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

```
% lb print helloworld person
??
```

ENTITY CONSTRUCTION WITH REFMODE

```
% lb addblock helloworld '
    person(p), person_name(p : name) -> string(name).'
```

```
% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

```
% lb print helloworld person
[3] "Sarah"
```

```
% lb exec helloworld '
    +person(p), +person_name(p : "Sarah").'
```

```
% lb print helloworld person
[3] "Sarah"
```

GET ANSWERS WITH LB COMMANDS

```
% lb predinfo helloworld person

info {
  name: "person"
  qualified_name: "person"
  arity: 1
  key_arity: 1
  value_arity: 0
  key_argument: "person"
  is_entity: true
  is_ref_mode: false
  ...
  has_ref_mode: true
  ref_mode_name: "person_name"
  derivation_type: EXTENSIONAL
  ...
}
```

GET ANSWERS WITH LB COMMANDS

```
% lb predinfo helloworld person_name

info {
  name: "person_name"
  qualified_name: "person_name"
  arity: 2
  key_arity: 1
  value_arity: 1

  is_entity: false
  is_ref_mode: true

  derivation_type: EXTENSIONAL
  ...
}
```

REFMODE CONVERSION

```
person_speaks(person, lang)
  -> person(person), language(lang).
```

REFMODE CONVERSION

```
person_speaks(person, lang)
  -> person(person), language(lang).

+person_speaks(p, l),
+person(p), +person_name(p, "Sarah"),
+language(l), +language_name(l, "English").
```

REFMODES XOR CONSTRUCTORS

- refmodes
  - one-to-one function (entity can have at most 1 refmode) : single value -> single value

    ```
    person(x), person_name(x : name) -> string(name).
    ```

  - EDB only
- constructors
  - More than one constructor predicate possible for the same entity type
  - EDB or IDB

CONSTRUCTORS

```
person(person) -> .

person_fromFirstLast[first, last] = person
  -> string(first), string(last), person(person).
lang:constructor(`person_fromFirstLast).
```

CONSTRUCTORS : EDB

```
person(person) -> .

person_fromFirstLast[first, last] = person
  -> string(first), string(last), person(person).
lang:constructor(`person_fromFirstLast).

% lb exec helloworld '
    +person(p),
    +person_fromFirstLast["Sarah", "Jones"] = p.'
```

CONSTRUCTORS : IDB

```
person(person) -> .

person_fromFirstLast[first, last] = person
  -> string(first), string(last), person(person).
lang:constructor(`person_fromFirstLast).
```

```
person(p),
person_fromFirstLast[first, last] = p
  <- input(first, last).

input(first, last) -> string(first), string(last).
lang:derivationType[`input] = "Extensional".
```

CONSTRUCTORS

```
person(person) -> .

person_fromFirstLast[first, last] = person
  -> string(first), string(last), person(person).
lang:constructor(`person_fromFirstLast).
```

```
person(p),
person_fromFirstLast[first, last] = p
  <- input(first, last).

input(first, last) -> string(first), string(last).
lang:derivationType[`input] = "Extensional".
```

```
% lb exec helloworld '+input("Sarah", "Jones").'
```

REFMODE VS. CONSTRUCTORS

```
person(person) -> .

person_name[name] = person
  -> string(name), person(person).
lang:constructor(`person_name).
```

```
person(person), person_name(person : name) -> string(name).
```

REFMODE VS. CONSTRUCTORS

```
person(person) -> .

person_name[name] = person
  -> string(name), person(person).
lang:constructor(`person_name).
```

- EDB or IDB
- multiple constructors
- no one-to-one guarantee

```
person(person), person_name(person : name) -> string(name).
```

- EDB only
- one refmode per entity
- one-to-one guarantee

SUMMARY OF CONCEPTS

- entities
  - user-defined types
  - leverage compiler to catch subtle bugs
- reference mode (refmode) predicates
  - provide references for entity values
  - one-to-one function from entity to refmode values
  - refmode values can be of any primitive type
  - allow construction of entity values using EDB rules

LESSON 3 LAB EXERCISES