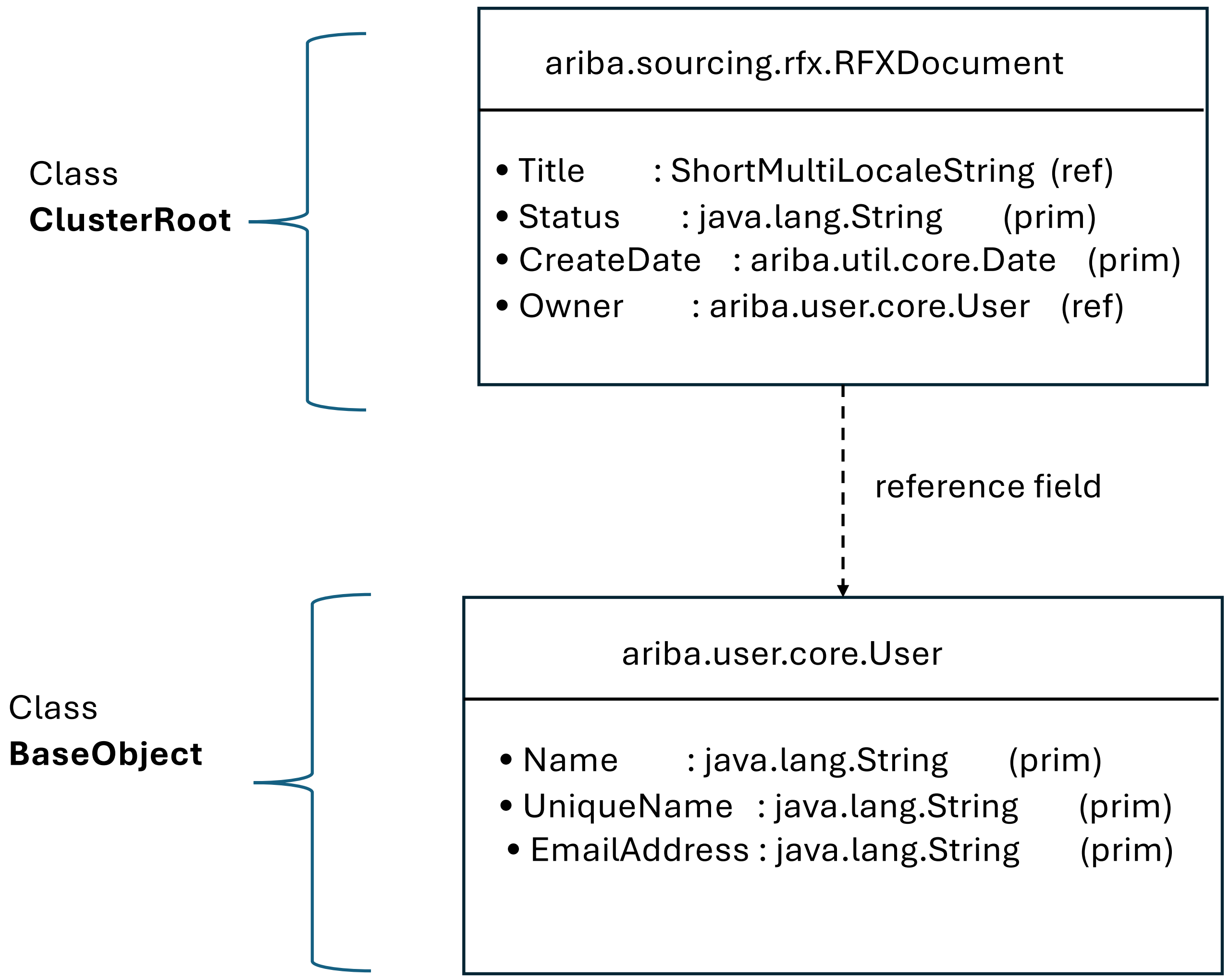
Ariba Query Language (AQL)

Hanieh Alipour

AQL

- A Java-level query layer with **SQL-like syntax** (SELECT … FROM … [WHERE …]) that lets us query using **object model classes & fields**, not DB tables/columns.
- It’s **integrated with metadata XML**, so queries are **strongly typed**, validated against our object model, and **insulated from physical schema changes** (AQL → SQL translation happens under the hood).



ariba.sourcing.rfx.RFXDocument

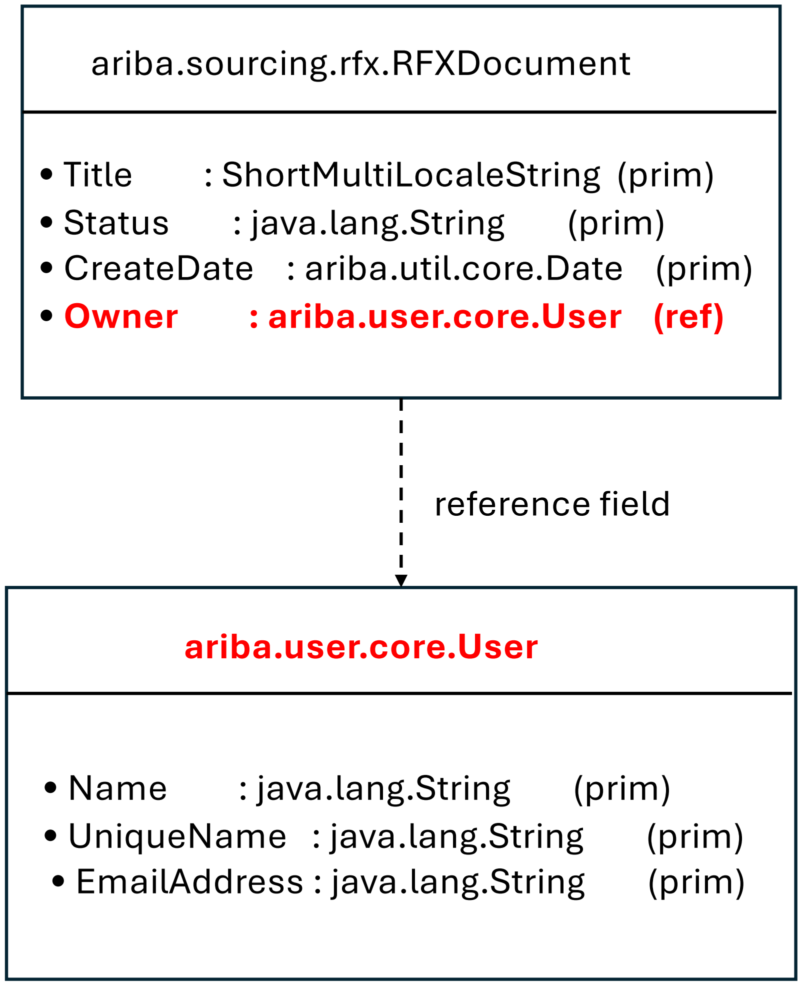
- Title : ShortMultiLocaleString (ref)
- Status : java.lang.String (prim)
- CreateDate : ariba.util.core.Date (prim)
- Owner : ariba.user.core.User (ref)

ariba.user.core.User

- Name : java.lang.String (prim)
- UniqueName : java.lang.String (prim)
- EmailAddress : java.lang.String (prim)

A class is a **ClusterRoot** if it **inherits from ariba.base.core.ClusterRoot** (directly or via parents).
Direct sign: super="ariba.base.core.ClusterRoot" on the class.

Select ------ From --------

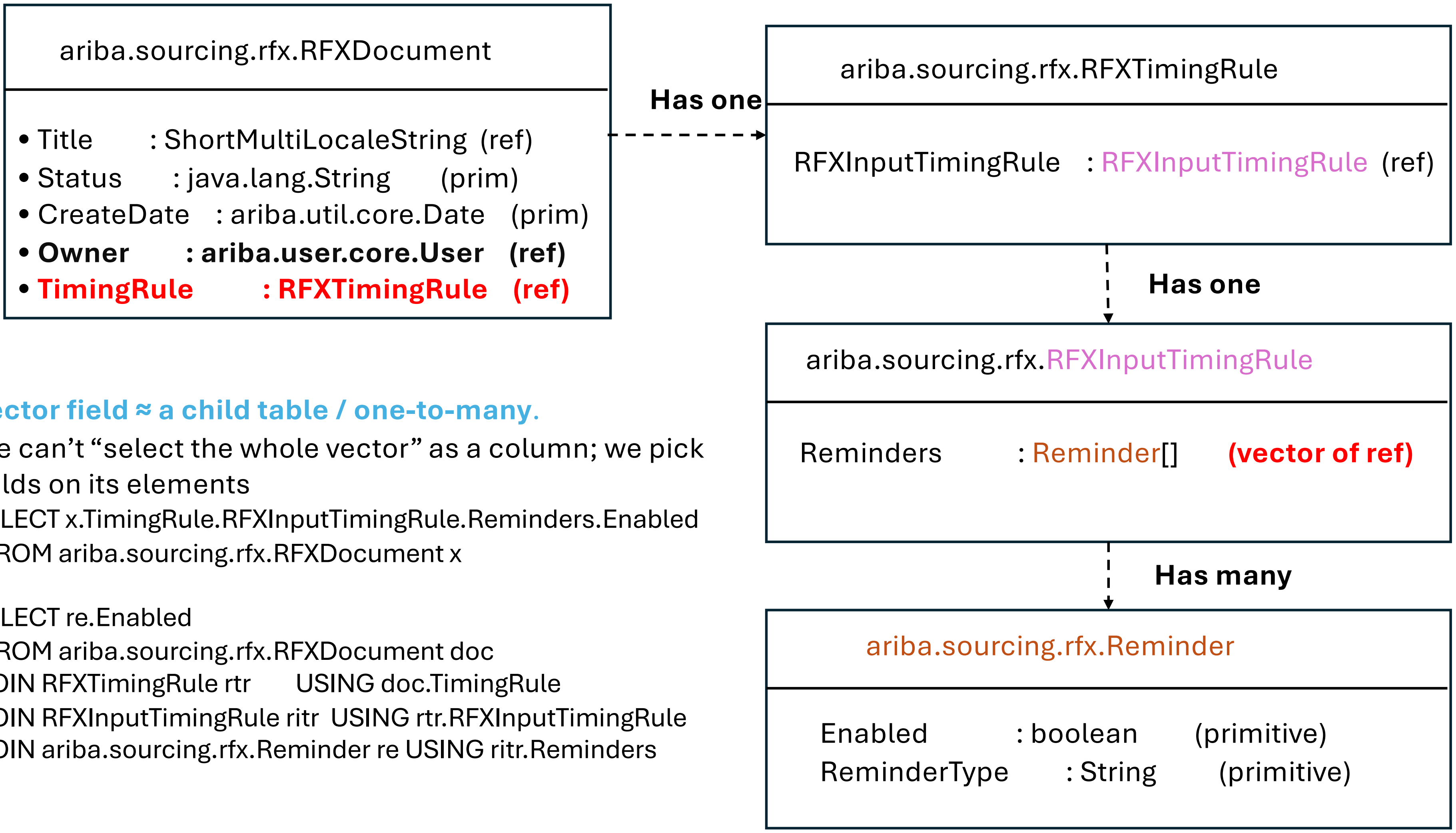

ariba.sourcing.rfx.RFXDocument
• Title : ShortMultiLocaleString (prim)
• Status : java.lang.String (prim)
• CreateDate : ariba.util.core.Date (prim)
• Owner : ariba.user.core.User (ref)
reference field
ariba.user.core.User
• Name : java.lang.String (prim)
• UniqueName : java.lang.String (prim)
• EmailAddress : java.lang.String (prim)

Field ≈ a column when it's a **primitive** (string, number, date, boolean).
Primitive fields = plain values
Examples: Status, CreateDate, DocumentVersion.
How to use: directly (no dot)

Reference field ≈ a foreign-key relationship
In AQL, we don't write JOIN … ON …; we **navigate** with dots (e.g., Owner.Name) or use AQL's JOIN … USING path syntax that references the relationship path, not an FK column.

```
SELECT x.Title, x.Owner.Name
FROM ariba.sourcing.rfx.RFXDocument x
```

```
SELECT u.Name, x.Title
FROM ariba.sourcing.rfx.RFXDocument x
JOIN ariba.user.core.User u USING x.Owner
```


ariba.sourcing.rfx.RFXDocument
• Title : ShortMultiLocaleString (ref)
• Status : java.lang.String (prim)
• CreateDate : ariba.util.core.Date (prim)
• Owner : ariba.user.core.User (ref)
• TimingRule : RFXTimingRule (ref)
Has one
ariba.sourcing.rfx.RFXTimingRule
RFXInputTimingRule : RFXInputTimingRule (ref)
Has one
ariba.sourcing.rfx.RFXInputTimingRule
Reminders : Reminder[] (vector of ref)
Has many
ariba.sourcing.rfx.Reminder
Enabled : boolean (primitive)
ReminderType : String (primitive)

Vector field ≈ a child table / one-to-many.

We can't "select the whole vector" as a column; we pick fields on its elements

```
SELECT x.TimingRule.RFXInputTimingRule.Reminders.Enabled
  FROM ariba.sourcing.rfx.RFXDocument x

SELECT re.Enabled
  FROM ariba.sourcing.rfx.RFXDocument doc
  JOIN RFXTimingRule rtr       USING doc.TimingRule
  JOIN RFXInputTimingRule ritr  USING rtr.RFXInputTimingRule
  JOIN ariba.sourcing.rfx.Reminder re USING ritr.Reminders
```

AML Inheritance — How it works (and why we care)

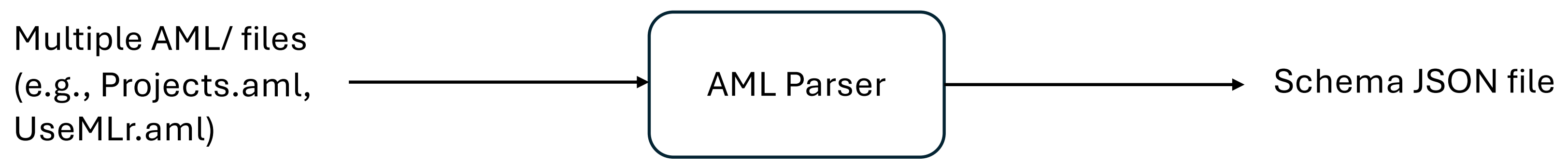
What it is?

- Classes in AML can **extend** another class via super="...".
- A child **inherits all fields** from its parent (and grand-parent, etc.).
- The child can **add new fields** or **override** parent fields (e.g., constraints).

Fine-tuning for AQL

Parser

- Our parser converts raw AML into a single, enriched schema JSON—complete with cluster roots, field kinds, display hints, vectors, and enums—so the generator can produce accurate, schema-aware NL→AQL pairs.



What it extracts per class?

- **Class identity & structure:** class, super, abstract, prefix, privacy.
- **Docs:** class_javadoc + field_javadoc (trimmed and preserved).
- **Indexes & lookup keys:** indexes, lookupKey.
- **Fields:** for each field we capture:
  type (primitive, reference class, or date)
  vector (is it a list)
  kind: "primitive" | "reference" | "vector"
  isDate (true for ariba.util.core.Date)
  refClass (for references)
  vectorOf ("primitive" or "reference")
  elemClass (element type if vector of references)
  validChoices (enum-like values, when present)
  plus AML attributes like nullable, indexed, privacy, aliasPath, typeAttrs

NL-> AQL Generator

- Takes the **parsed AML schema.**
- For each class, builds a **mini schema slice** the model can safely use.
- Asks an LLM to write **business questions → AQL** pairs **that only use that slice**.
- **Validates & auto-repairs** the AQL (dates, dotted fields), then saves clean pairs.
- Writes each example as { input, schema, output } so the schema travels with the data.