IceDust 2: Composition of Calculation Strategies and Multiplicity Bounds on Derived Bidirectional Relations

Daco C. Harkes and Eelco Visser
Delft University of Technology, The Netherlands
d.c.harkes, e.visser@tudelft.nl

Declarative Data Modeling

Derived values are values calculated from base values. A key concern in implementing systems with derived values is minimizing the computational effort that is spent to re-compute derived values after updates to base values. A key concern in modeling systems with derived values is minimizing the programming effort to realize such minimal computations. For example a calculation strategy may only depend on fields with the same or a stronger calculation strategy. The grey box below to the right shows the lattice of calculation strategies, lower is stronger.

Calculation Strategies

IceDust 2 provides three strategies for calculating the derived values: on-demand, incremental, and eventual. The distinction between these strategies is when derived values are calculated:

- on-demand calculation
- incremental calculation
- eventual calculation

Derived Relations

Derived values can be expressed with views in relational databases, but they do not provide multiplicity bounds. Derived values can also be expressed with expressions in incremental or reactive programming, but require significant work to encode bidirectional derived values. Derived relations are specified as expressions in IceDust 2, this provides multiplicity bounds. Bidirectionality is provided by maintaining inverses.

Example Data

Alice passes the course, her grades are sufficient, and the lab is handed in on time. Bob’s exam grade is insufficient. Bob’s lab is late, but he received a pass for the personal deadline for the course.

Sound Composition of Calculation Strategies

Calculation strategies should retain correctness and time complexity under composition. For example a read of an incremental value is O(1), as such it cannot reference an on-demand value, as it would have to check whether it changed.

Example IceDust 2 Specification

entity Assignment (eventual) {
  name : String
  value : Base Value
  question : String?
  deadline : Datetime?
  finished : Boolean
  avgGrade : Float = avg(submissions.grade) Derived Value
  passPerc : Float = count(submissions.filter(x => x.pass)) / count(submissions)
}

entity Student {
  name : String
}

bidirectional relation Submission.student 1 <--> * Student.submissions Bidirectional Relation
relation Submission.assignment 1 <--> * Assignment.submissions relation Submission.parent ? <--> * Assignment.children
relation Submission.parent ? = assignment.parent.submissions.find(x => x.student = student) relation Submission.children

Bidirectional Relations

When updating bidirectional relations, both multiplicity and bidirectionality have to be preserved. Multiplicities guide bidirectional updates in IceDust 2. For example executing lab.addToChildren(exam) implicitly removes math as parent from exam, as exam can at most have one parent. It is identical to executing exam.set(Exam) with Exam.children = Exam.children.add(lab).

Derived Bidirectional Relations

Updating derived values might lead to bidirectional maintenance, which in turn can lead again to updating derived values. For example when executing exam.set(Exam) with Exam.children = Exam.children.add(lab). This causes the deadlines of those objects to be updated, moreover, MathCourse.children, lab.Alice.children, mathBob.children, and labBob.children are also updated, which triggers re-execution of the grade for those objects.

Harkes, D. C., Visser, E.: Underying and Generating Relations in Rule-Based Data Modelling and Navigation. SLE (2016)