Relational Data Model

SOFTWARE GEMS

Purpose

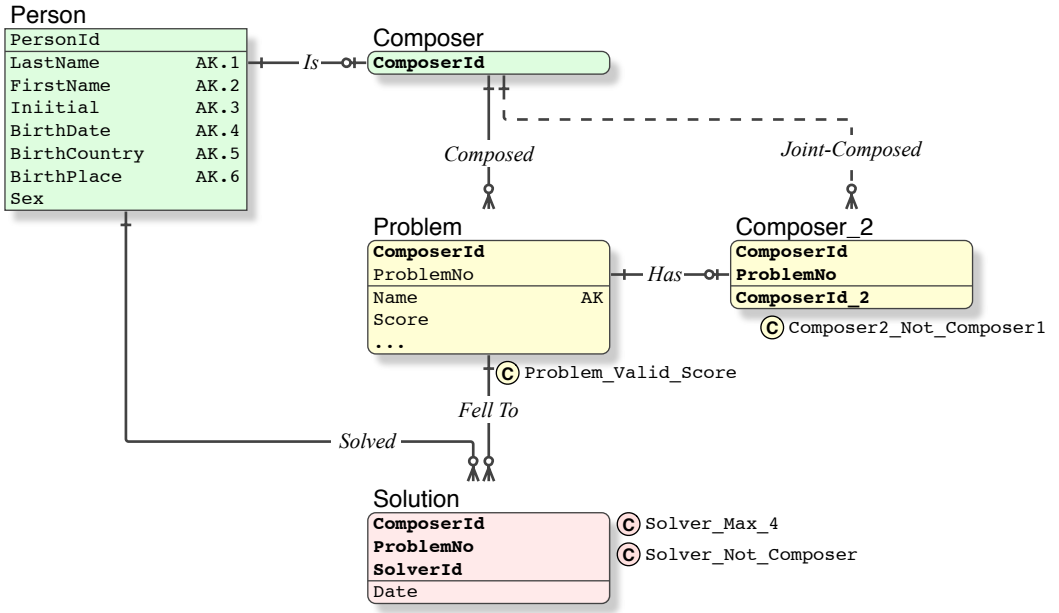
This is a response to the droolers at *Hey Presto, The Torrid Manifesto*, who hilariously claim that the *Relational Model can't do this*, or *SQL can't do that*. Quite oblivious to the fact that both are founded on First Order Predicate Calculus, which has no limit.

The alleged "impossible in the *RM*" is solved by the ordinary knowledge of the *RM*.

The alleged "impossible in the SQL" is solved by the ordinary knowledge of the SQL, and a genuine SQL Platform.

As of 2007, SQL allows a `CONSTRAINT` to call a Function, which of course has the scope of the set or the database.

The requirement was to fulfil those discussed at *TTM*. No suggestion that it (the particular model with weird constraints) is correct for any other purpose.



Composer
- Composer is a subset of Person
- Problems are created by a Composer, not any Person.

Problem
- The notion of an Independent Problem (one that exists without a Composer to create it) is absurd
- The use of surrogates on initially-perceived entities cripples the modelling process: Problem is not an Independent fact
- Thus Problem is **Identified** by its ComposerId, and a sequence of ProblemNo within that
- The Alternate Key(Name) ensures that each Problem is unique.

Composer_2
- Somewhat after the discussion, the model was completed to fulfil the full original requirement. Since the second Composer is "rare", an Associative table for Composers (plus a Constraint that calls a Function that limits it to two Composers) is not reasonable.
- A simple `CHECK` Constraint is required to ensure the Composers are not the same.

Solution
- The ComposerId and the SolverId now appear in each row
- Thus a simple `CHECK` Constraint can be used, and the Function (required for checking other rows) is eliminated

Entity Type
- Identifying Entity
- Transaction
- TransactionDetail

DDL

IDEF1X Notation

Blank