Ceda Constraints

David Barrett-Lennard

16 Oct 2016

# Data integrity

Constraints are great for ensuring data integrity.



But there are some issues with imposing constraints which we will discuss in this document.

# Constraints ⇒ OT is intractable

There is a solution for OT on text, where the only constraint is that the characters are ordered. Now consider that we impose a much more complex constraint, say that the string is a valid Java program according to the Java language specification (e.g. it much conform to the grammar). It is difficult (impossible?) to ensure that in general the merge of arbitrary concurrent operations will satisfy such a constraint.

Text

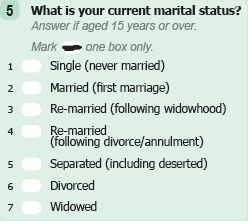


Java

???

# Constraints ⇒ can't record partial information

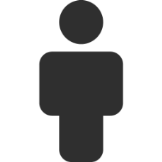
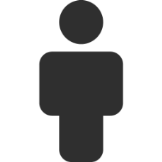
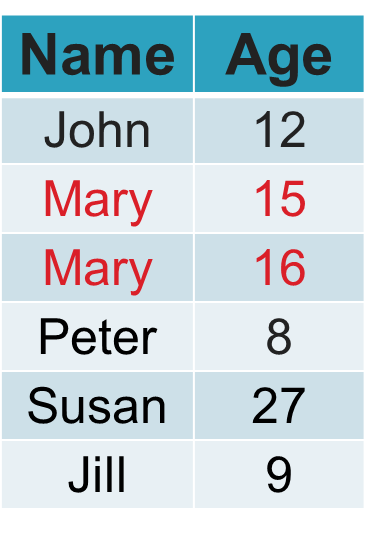
"Mark one box only" looks like an integrity constraint. But if we impose it we end up recording less information. The real world is messy; we often need relaxed constraints in order to record partial information about the world.



Integrity constraint

# Constraints ⇒ can't record conflicts

Relaxing a key constraint allows for recording conflicting information. That means the DBMS can merge edits and keep the conflicting information, allowing the users to fix it up as appropriate.



Mary is 15

Mary is 16

Key constraint ?

# Constraints ⇒ hide problems

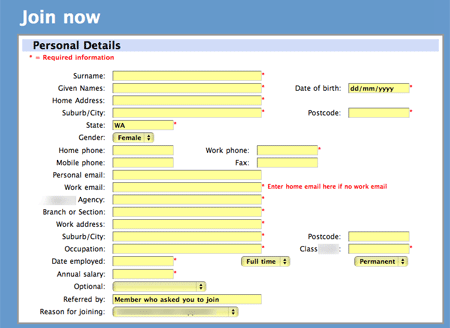
The reality is that information entered into a database is imperfect because users are not omniscient and they make mistakes. Also updates are asynchronous which can create temporary conflicts.

Imposing integrity constraints can mean the data always looks perfect, giving a false sense of security. The database provides no information about how often problems are occurring if it doesn't record conflicting or missing information.



# Constraints ⇒ drafts not managed by DBMS

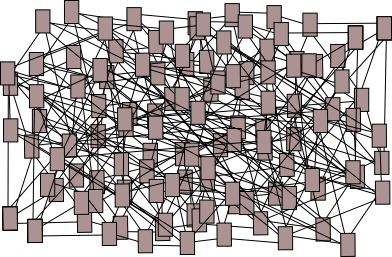
Consider all those times you have filled out a form and something goes wrong and you have to fill it out from scratch again. It is better for database management systems to manage all data (including data as it is entered – i.e. drafts).



**\*** = integrity constraint

# Constraints ⇒ complex updates

Constraints tend to mean multiple updates must be made to the data, it is not possible to update variables independently.



# Constraints ⇒ fallible updates

Constraints can mean an innocuous looking assignment to a simple variable raises an integrity constraint violation exception

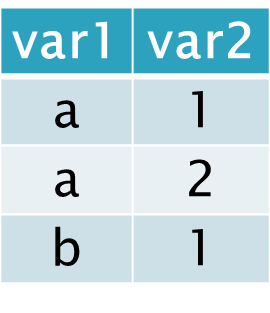


**x = 10; // throws exception?**

# Constraints ⇒ risk deadlocks

A wait-for graph is a directed graph used for deadlock detection in operating systems and database systems.

In the following example, Txn1 locks var1 then var2. Tsn2 locks var2 then var1. This can lead to a cycle in the wait-for graph, a dead-lock. An integrity constraint over both var1,var2 makes this more likely, because any update to either variable will require reading the other to check the constraint.



var1

var2

Txn1

var1 = 'b';

CheckConstraint();

Txn2

var2 = 2;

CheckConstraint();

Txn1

Txn2

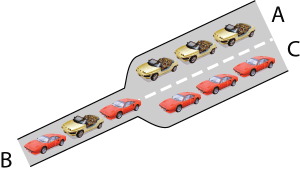
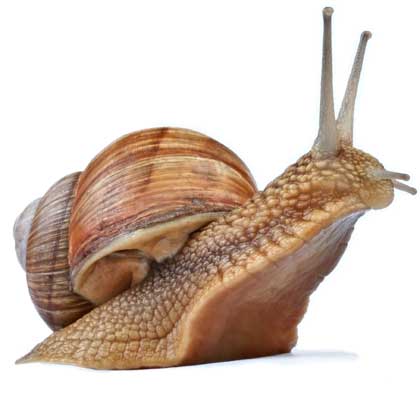
Cycle in wait-for graph

Constraints on (var1,var2)

# Constraints ⇒ slow updates

Constraints slow down updates in three ways:

1. The overheads of the constraint checking itself
2. The reduction in concurrency because variables cannot be updates independently
3. Detection and handling of dead-locks (e.g. selecting a victim and aborting it)



# Summary: the pros and cons of constraints

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Forces users to be careful * Avoids errors in the data * Data model is simpler * Queries are simpler | * Less concurrency * Less logical independence * More dead-locks * Poor writer performance * Updates are complex * Updates are fallible * Need compensating actions * OT is intractable * Draft data not managed * Cannot record conflicting information * Unrealistic   + Users are not omniscient   + Users make mistakes   + Updates are asynchronous * False sense of security |

# Dealing with integrity constraints

There is a solution where we get our cake and eat it too (i.e. all the advantages without the disadvantages). We apply constraints *indirectly*.

Derived variables

Base

variables

Unconstrained

Constrained

Non-injective function

Writers

Readers



Writers access a base representation which is relatively unconstrained. A function maps the base representation to a derived representation which imposes the constraints.

Multi Version Concurrency Control (MVCC) is very useful for allowing concurrency between readers and writers.

Satisfying a complicated constraint is made easy, rather than it being difficult to achieve. Users are able to drive the output through a complicated space.

Update operators tend to be simple and infallible.

The function mapping to read only derived views may not be injective. That means that the base representation isn’t canonical. There can be significant advantages to not imposing canonical representations. Canonical representations often seem arbitrary (because they break symmetries). E.g. represent a triangle as an array of three vertices, but triangles have invariant under cyclic permutations of their vertices.

# Non-canonical representations

Non-injective function

Base

Derived

Representations

Represented value

# Representation of a rational number with strong constraints

It is quite common to impose a canonical representation constraint on a representation. That often means a partial constructor function. In many cases it is possible to relax the constraint (think of any member of the equivalence class as being available to represent the class, and implement an equality operator to account for the fact that the representation is non canonical)

No Cartesian factorisation

1

2

4

5

6

7

8

3

1

2

4

5

6

7

8

3

bijective

(n,d)

Representation

Constraint:

(n,d) ∈ Z2 &

d>0 &

(n=0 & d=1 or coprime(|n|,d))

↔

n

d

Rational number

# Representation of a rational number with weak constraints

We can weaken the constraint so that the representation is non-canonical. For example there are multiple representations of the rational number 2/3.

Cartesian product

Non-injective function

(n,d)

Representation

Constraint

(n,d) ∈ Z × Z\{0}

n

d

Rational number

(2,3)

(4,6)

2

3