Managing integrity constraints on relational databases

Renaud De Landtsheer
Personal Motivation

- Hobby based on my personal background
  - Formal methods, development of formal tools
  - Directed towards classical SQL-based systems
- Not fundamental research
  - [Hesh84, Hsu85, Lloyd87, Sardi88, Qia88, Cha90, Deck94, Lee96, Selj99, Deck02, Mar05]
- Just target a robust enough implementation
  - Something that can be executed without requiring any form guidance or application-specific tuning
State of practice in DB development

- Integrity constraints are critical for high assurance systems (banking, state, etc)
- In general, integrity is handled by triggers
- Writing triggers by hand
  - Is difficult, error prone
  - Takes time, especially for complex constraints
  - Requires high expertise
- No automated solution available in DBMS, except for simple constraints (e.g.: foreign keys, single row constraints, type constraints)
  - We might cover eg. the patterns of [Hud10]
Approach [Mart05, etc.]

- Automatically generates integrity triggers from declarative integrity constraints

What it gives
- An improvement of quality
- A gain in time

What it costs
- Writing the integrity constraints in SQL-like notation
- Running the tool
Bad approach: querying integrity

- Each integrity constraint can be queried at each modification on database
- One can relatively easily generate the triggers implementing this approach
- This approach can be unnecessarily costly
  - Constraint was true before modification, modification only touches part of the domain, why check everything?
  - Some constraints might overlap each other, how about taking into account this overlap to speed up verification?
Given
- Database schema
- Integrity constraints

Automatically generate
- Triggers to be embedded into the schema

Such that
- Events violating the integrity are blocked by the triggers
- Triggers have low overhead
How to ensure a low overhead of integrity checking?

- Triggers perform **incremental** verification
  - The constraint is supposed to be true before the event
  - Only verify the constraint on the part of data that has changed

- **Take into account other** integrity constraints
  - They might partially overlap

- Use of **two-step** checking
  - A first local check, sufficient, but not necessary
    - This is intended to filter out some of the updates
  - A second global check in case the first fails, possibly on the whole DB
    - Knowing that the local check failed
An example

- **Input**
  - Schema

<table>
<thead>
<tr>
<th>Referencer</th>
<th>Referenced</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForeignKey: Int</td>
<td>PrimaryKey: Int</td>
</tr>
<tr>
<td>OtherField: Int</td>
<td></td>
</tr>
</tbody>
</table>

- **Integrity constraint**

  For all \( r : \text{Referencer} \)

  \[ r.\text{OtherField} = r.\text{ForeingKey} \]

  OR exists \( d : \text{Referenced} \)

  \[ r.\text{ForeignKey} = d.\text{PrimaryKey} \]

- **Output:** Five triggers monitoring events of the DB

  - INSERT(Referencer)
  - UPDATE(Referenced, PrimaryKey)
  - DELETE(Referenced)
  - UPDATE(Referencer, ForeignKey)
  - UPDATE(Referencer, OtherField)
Example: detail of one trigger

CREATE TRIGGER CHECK_INSERT_ONREFERENCER

BEFORE INSERT ON Referencer
FOR EACH ROW

If NOT NEW.OtherField = NEW.ForeignKey THEN

If NOT Exists d:Referenced
    d.Primarykey=NEW.ForeignKey THEN

    FAIL
    END

END

This is the output of my prototype
Example: detail of another trigger

```
CREATE TRIGGER CHECK_UPDATE_ONREFERENCER_OTHERFIELD

BEFORE UPDATE ON Referencer.OtherField
FOR EACH ROW

If NOT(
OLD.ForeignKey = NEW.OtherField
OR OLD.ForeignKey != OLD.OtherField
OR OLD.OtherField = NEW.OtherField)
THEN

If NOT Exists d:Referenced
  d.Primarykey=OLD.ForeignKey
THEN
  FAIL

END

This is the output of my prototype
```
First order as constraint language

Assertion ::= Exists variable:Table Assertion
| ForAll variable:Table Assertion
| Assertion AND Assertion
| Assertion OR Assertion
| NOT Assertion
| Assertion Implies Assertion
| Expression = Expression
| True
| False

Expression ::= Constant
| Variable.Column
In the box

Integrity constraints

Event Identifier

Time Shifter

Constraint before

Event

Constraint after

Database schema

Axiom generator

Primary keys

Simplifier

Conditions
Example of time shifting

- Shifting problem:
  - Assertion: Forall x:X P(x)
  - Event: Insert(x)
  - Evaluated on the database before the modification

- Assertion before:
  - Forall x:X P(x)

- Assertion after:
  - (Forall x:X P(x)) AND P(new)
Key component: the simplifier

- My implementation:
  - BDD-based simplification on skolemized assertions
  - Borrows from abduction procedures and from PVS
- Greedy approach: simplify from left to right when visiting the assertion to simplify
- Limitation:
  - Some axioms are very costly (transitivity of equality), so need for heuristic simplification of axioms, based on assertions
  - Would be better to simplify according to the amount of data expected in each table
Targeted Look, integrated into graphical database development software

Referencer
- ForeignKey: Int
- OtherField: Int

Referenced
- PrimaryKey: Int

Conditional ForeignKey

For all \( r: \text{Referencer} \)

\( r.\text{OtherField} = r.\text{ForeignKey} \)

OR

Exists d: Referenced

\( r.\text{ForeignKey} = d.\text{PrimaryKey} \)
My targeted contribution

- Generate both local and global check (TTM inspiration)
- An implementation
  - What I have
    - 80% of condition generator
    - Conditions double-checked by automated theorem prover (Prover9)
  - SCALA implementation
Long term vision for fundamental research?

- This technique of integrity checking **discovers** updates to DB

- Better 1: Annotating values with their origin
  - For instance, when one inserts a row with a foreign key, one generally queries the referenced primary key before inserting
  - How about keeping track of the origin of the value set to the foreign key?

- Better 2: Adding condition checking to business code
  - From server-side verification to client-side verification
  - The technology advertised here could be used for this
Bibliography

- [Mart05] Davide Martinenghi, Advanced Techniques for Efficient Data Integrity Checking Ph.D. Dissertation, October 2005
- [Hud10] Adrian Hudnott, What are the Most Common Forms of Database Integrity Constraint?, 16th February 2010, tech report