Automated Design of Updateable Database Views: a Framework for Possible Strategies

Stephen J. Hegner
Umeå University
Department of Computing Science
SE-901 87 Umeå, Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Disclaimer and (Modest) Goal

- My field of expertise is neither conceptual design nor automated reasoning.

So within the context defined by those fields, I probably do not know what I am talking about.

- My interest is in *database views*, in particular:

  Constraints: Characterize the constraints on a view, given the constraints on the main schema.

  Updates: Develop ways and understand how updates to views may be supported in a systematic fashion.

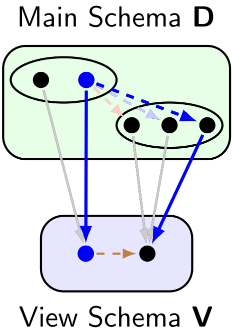
  Structure: Understand how views interact with one another, and more generally their mathematical properties as a collection.

Goal: In this short presentation, some ideas of the problems which (from my limited perspective) must be addressed in order to perform conceptual design of schemata with updateable views will be identified.

Views and the View-Update Problem

Views: A *view* of a schema **D** provides partial information about the state of **D**.

- The underlying mapping is usually defined by a quotient operation in which each view state corresponds to an equivalence class of states of **D**.
- This means that a given view update will have many *translations* to the main schema (it will always have at least one).

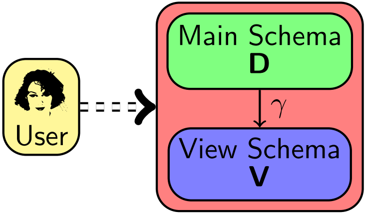


- The *view-update problem* is to determine:
  - which reflections, if any, are suitable; and
  - if there is more than one suitable choice, which is best.
- The view update problem is a *design* problem with no universal answer.
- However there are principles to be considered.

Open vs. Closed Views

Open view: The user has access to both the view schema and the main schema.

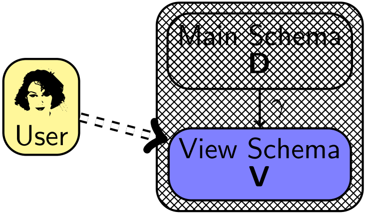
- The view is thus a “helper”.
- The user has enough information to define the update translations herself.



Open vs. Closed Views

Open view: The user has access to both the view schema and the main schema.

- The view is thus a "helper".
- The user has enough information to define the update translations herself.



Closed view: The user sees only the view.

- The user has no direct knowledge of the base schema.
- The view must be self contained in terms of knowledge needed to effect updates.
- The view should look "just" like a complete base schema.

Focus: The view design issues addressed in this talk will focus upon closed views.

Simple Views with Complex Constraints

Goal: To present a closed view, it is highly desirable to be able to describe the integrity constraints in a simple way.

Problem: Unfortunately, this is often not possible.

Example: There is a relational schema $R[ABCD]$ with three FDs, for which the constraints on the projection Π_{ABC} are not finitely representable.

- $\mathcal{F} = \{A \to D, B \to D, CD \to A\}$.

Example: There is a relational schema $R[AB]$ with one FD for which the constraints on the pair of projections (Π_A, Π_B), regarded as a view, are not first order (for infinite databases).

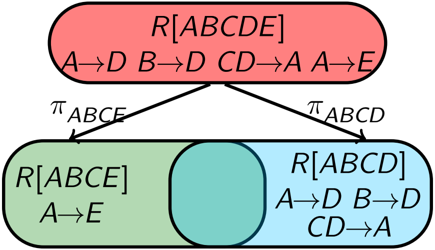
- $\mathcal{F} = \{A \to B\}$.
- Constraint $=$ Card$(B) \leq$ Card(A).

Conclusion: It is not always realistic to provide a full characterization of the constraints on a view.

Solution: Limit the allowable updates, and provide only constraints which are necessary to define acceptable updates.

Localization via Constant Complement

- In general, a view update has many translations to an update on the main schema.
- The best choice may be formalized via *localization*.



Localization: Restrict the reflected changes to the main schema to that part which corresponds to the view.

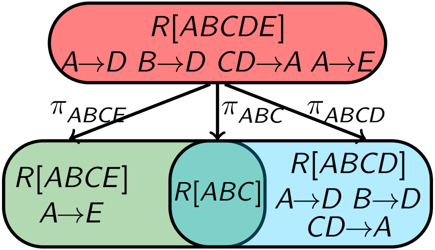
Example: To update the view Π_{ABCE}, change only $ABCE$ values in $R[ABCDE]$.

- Keep the *complementary* part constant.
- In this case, a complement is Π_{ABCD}.

Why require a complement? It defines a lossless decomposition, so it determines unambiguously how the update is to be translated.

Localization via Constant Complement – State Invariance

State invariance: The admissibility of a view update must depend upon the view state only; not upon the state of the main schema.



- Guaranteed if the view and its complement form a dependency-preserving decomposition (*meet complements*).
- A view update is allowed if:
  - The embedded constraints are satisfied.
  - The common view (*meet*) is held constant.
- In the above example, the allowed updates to Π_{ABCE} are those which satisfy the FDs and keep the meet Π_{ABC} constant.
- This holds even though the view Π_{ABCE} is not finitely axiomatizable.
- For meet complements, the view axioms which need to be satisfied by valid updates are no more complex than those of the main schema.

Localization via Constant Complement – Other Invariance

- There are two other forms of invariance which are important in the design process.

Problem: The translation of a view update may depend upon the choice of complement.

Solution: *Reflection invariance* is guaranteed for insertions and deletions if the view mappings are monotonic.

  - It may be guaranteed for other updates as well, but pathological exceptions exist.

Problem: There need not be a *universal* complement which supports all updates which are supported by some complement.

- Simple counterexamples exist to *update-set invariance*.
- There is no widely applicable solution to this problem.
- Often, a maximal set of view updates to be supported must be chosen in the design process.

View Specification

Question: How should a view be specified in the design process?

Proposal: The following information is necessary:

- Information content of the view;
- View updates to be supported.

- The information-content issue is a bit more complex.
- To support the given updates via a suitable constant-complement strategy, it may be necessary to include more than the given information content.
  - More information makes it possible to find a smaller complement, and thus a better chance of supporting all of the updates.
- On the other hand, if no bound on the allowed information in the view is given, then the identity view gives a trivial but probably not very useful solution.
- The following refinement on information content of the view is proposed:
  - Minimal information content of the view;
  - Maximal information content of the view.

Automation of Updateable View Design

Context: A set $\mathcal{V}$ of views which includes both the possibilities for the view to be updated and the candidate complements.

- The search process must not find only a complement to the view to be updated, but within the min-max constraints, that view itself.

Algorithm: The algorithm must identify suitable pairs $(\Gamma, \Gamma') \in \mathcal{V} \times \mathcal{V}$ in which Γ is the view to be updated and Γ' is a suitable meet complement.

- The updates must not change the state of Γ'.
- So a bigger Γ might allow a smaller Γ', with a greater chance of success.

- Recall that meet complements are characterized by lossless and dependency-preserving decompositions.
- Testing for losslessness is relatively easy in many settings.

Embedded covers: The key to success for any such algorithm is thus the ability to test for and analyze embedded covers.

- Determining whether or not a pair of projections on a universal relation constrained by FDs has an embedded cover is NP-complete.
- Thus, algorithms which are worst-case tractable (in the formal sense) are essentially ruled out.
- However, there may still be many situations in which solutions may be found effectively for many practical cases.

Suggested context for investigation:

Constraints: FDs and simple inclusion and cardinality constraints.

- More general constraints could be allowed, as long as the view to updated does not “split” those constraints.

Views: The equivalent of *SP*-views, with projection and selection.

- Enough is known about this context that some useful results could likely be obtained (with some work).

Conclusions and Further Directions

Conclusions:

- There appear to be fertile areas for investigation of automated (updateable) view design based upon the constant-complement strategy.
- These are contingent upon suitable algorithms for finding embedded covers of the class of dependencies considered, into the class of views considered.

Further Directions:

- Explore algorithms for finding embedded covers efficiently.
- Apply these algorithms to the problem of automated view updateable view design.

A Request:

- If there has been work on the conceptual design of (updateable) views, I would very much appreciate some pointers to the work.