Carleton
UNIVERSITY

Evolution and Change in Relational Databases

An Overview of Some Issues

Leopoldo Bertossi[⋆]

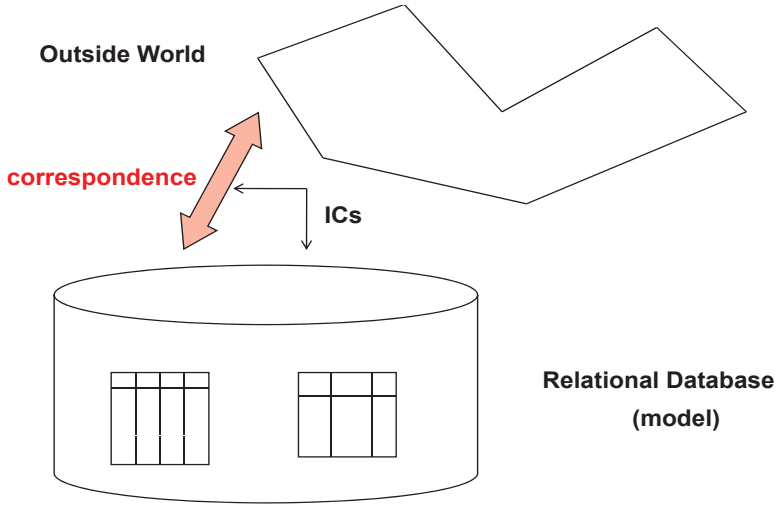
Carleton University
School of Computer Science
Ottawa, Canada

⋆: Faculty Fellow of the IBM Center for Advanced Studies

Topics:

1. Relational DBs and ICs
2. Database and View Maintenance
3. Incremental Approaches
4. ICs on Views
5. Updates through Views
6. Inconsistency Handling
7. Specifying Change

1. Databases and Integrity Constraints



A database instance D is a model of an outside reality

An integrity constraint on D is a condition that D is expected to satisfy in order to capture the semantics of the application domain

A set IC of integrity constraints (ICs) helps specify/maintain the correspondence between D and that reality

Several applications: [Godfrey et al. 98]

- By being satisfied, an IC imposes a restriction on the evolution of the DB under updates
- When guaranteed to be satisfied, ICs can be used for semantic query optimization
- ICs properly represented become metadata, i.e. data about the data Many uses, e.g. interoperability

IC enforcement:

- By the DBMS itself when IC have been declared with schema (limited applicability)
- Through triggers (active, ECA rules) stored by user in DB

  Reject/notify inadmissible updates or compensate updates

  Can be derived from ICs [Widom et al. 95]
- Through application/transaction programs

Logically?

A relational DB D can be seen as a set-theoretic structure

An IC as a sentence φ in language associated to DB schema

So, $D \models \varphi$ is well defined as model-theoretic satisfaction in FO predicate logic

Alternatively, D can be seen as a theory $Th(D)$ written in FO predicate logic: Reiter's logical reconstruction [Reiter 84]

Manager	Boss	Subordinate
	ken	$john$
	$john$	$mary$
	$peter$	joe

- Predicate extensions plus CWA:

  $\forall xy(Manager(x,y) \leftrightarrow x = ken \wedge y = john \vee \cdots \vee x = peter \wedge y = joe)$

- Possibly domain-closure: $\forall x(x = ken \vee \cdots \vee x = mary \vee x = sue)$
- UNA: $john \neq mary$, etc.

Now $D \models \varphi$ also makes sense as $Th(D) \vdash \varphi$

ICs have to be entailed by the DB ...

Contrast with more recent views of DB, e.g. for an incomplete DB D and a set Σ of ICs: [Greco et al. 12]

- For D seen as a structure (closed set of ground atoms), maybe $D \not\models \Sigma$
- However, it is good enough if $D \cup \Sigma$ is consistent
- D is usually extended through Σ (chase, etc.)

Incarnation of older discussion: [Reiter 92]

- Reiter: ICs are satisfied by the DB
- Kowalski: ICs are to be consistent with DB

2. Database and View Maintenance

Database maintenance is about keeping the ICs satisfied when the DB undergoes updates

View maintenance is about keeping materialized view extensions synchronized with base table

A view is just a (usually virtual) relation defined on top of base (usually material) relations

A view defined on table $Manager$ (cf. page 5):

$$\forall x(TopBoss(x) \leftrightarrow \exists y Manager(x,y) \wedge \neg\exists z Manager(z,x)) \qquad (*)$$

$TopBoss(D) = \{ken, peter\}$ If $\langle sue, ken\rangle \mapsto D$?

The two problems are related ...

- Associate a violation view V_φ to IC φ

  $$\varphi \text{ is satisfied by } D \text{ iff } V_\varphi(D) = \emptyset$$

  Example: $FD: Subordinate \rightarrow Boss$ on previous schema

  $$V_{FD}(x, y) \leftrightarrow Manager(x, y) \wedge \exists z(Manager(z, y) \wedge x \neq z)$$

  To maintain the IC (satisfied), maintain the violation view (empty)

  BTW, a condition that is commonly used by ECA rules for IC maintenance

- A view definition, e.g. (*), can be seen as an IC expressed in an expanded language

  The definition has to be kept satisfied

Techniques for each of database and view maintenance can be applied to the other problem

In both cases, incremental techniques are desirable

3. Incremental Approaches

The issues:

- We do not want to check the full IC every time the DB is updated

Maybe an update on base table is irrelevant to the IC

Maybe only "a portion" of the IC has to be rechecked

- We do not want to recompute the view from scratch using the definition every time base tables are updated

Crucial for materialized views, as in DWHs [Gupta et al. 99]

Maybe the update is irrelevant to the view (definition)

Maybe it is a matter of computing a delta

Hopefully without the whole DB

"Inductive" IC checking: [Nicolas 82]

1. Assume $D \models \varphi$
2. Update D into D' by a set U of updates
3. What portion of φ (if any) has to be checked on D' (or only D, U)?

Example: With FD: $Subordinate \rightarrow Boss$ above

Assume $D \models FD$ Instead of checking on D':

$$\forall xyz(Manager(x, y) \wedge Manager(z, y) \rightarrow x = z)$$

1. If U contains only deletions:

   Do not check anything

2. If U is $insert_{Manager}(a, b)$:

   Check on D: $\exists x(Manager(x, b) \wedge x \neq a)$?

General mechanism that relies on syntactic structure of ICs

View maintenance and relevant updates: [Blakeley et al. 89] [Gupta et al. 95]

- Self-maintainable views: View update without accessing base tables, but instead
  - current, material state of the view
  - view definition
  - the actual updates
  - possibly ICs on base tables

No need for the updated underlying base data ...

Example: (the gist) Schema $R(A, B)$ with $FD: \ B \to A$

Instance $D = \{R(a, b), R(c, d), R(e, f)\}$

View (projection of R on B): $V(Y) \leftarrow R(X, Y)$

$V(D) = \{b, d, f\}$

With update $delete_R(a, b)$, using

- the update itself (knowing it has an effect on V)
- the pre-update extension of the view
- the FD (assumed to be satisfied so far)

we obtain right away the new extension: $V(D') = \{d, f\}$

- Irrelevant Updates: Determine views that are not affected by certain classes of updates on base tables

Ignore those updates for view maintenance

The "irrelevant update problem"

Example: (as above, cont.) For the view $V(X) \leftarrow R(X, Y)$

Updates of the form $change_{R[Y]}(\bar{t}; v)$ (in tuple $\bar{t}$ in R change value for Y into v) are always irrelevant

The irrelevant update problem also appears in IC maintenance: some updates never lead to inconsistency

For example, for FDs tuple deletions are always irrelevant

4. ICs on Views

Having ICs on views could be useful for the tasks above, for monitoring the DB behaviour through the views, query answering using views, metadata for interoperability in general ...

The classic problem of deriving ICs for views from view definitions and ICs on base tables [Klug 80, 82]

Example:

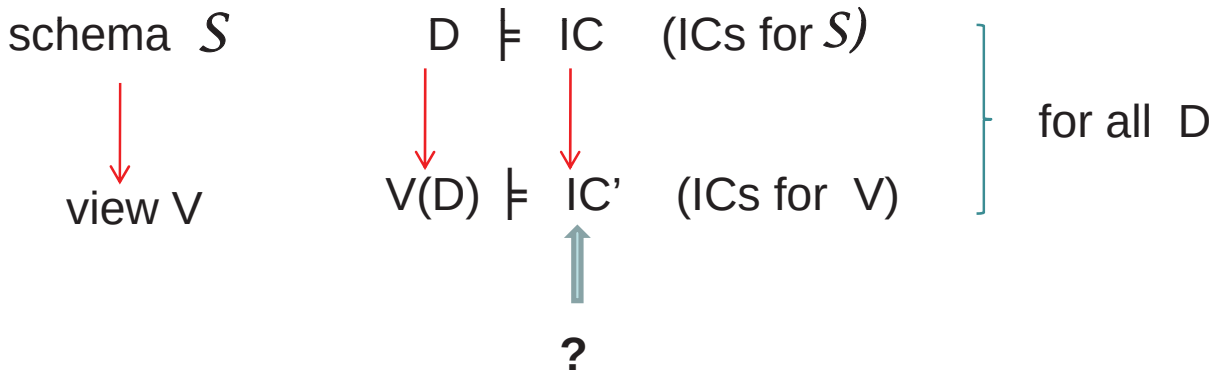
Manager	Boss	Subordinate	Salary
	ken	$john$	100
	$john$	$mary$	120
	$peter$	joe	150

FD: $Subordinate \rightarrow Boss$

View $V(x, y)$: $\exists z Manager(x, y, z)$

From V's definition and FD, an IC on V: FD^V: $Subordinate \rightarrow Boss$

Any violation of FD^V indicates a violation of FD

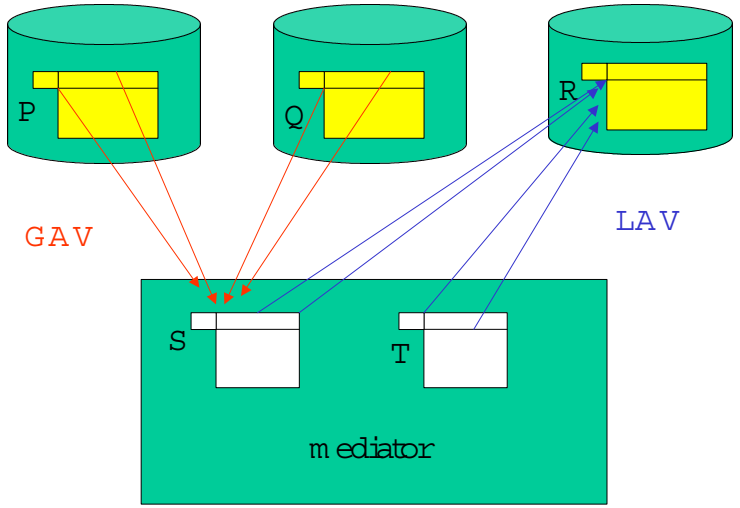


There are syntactic techniques for deriving the ICs on the views

However, in virtual data integration, under GAV:

Global ICs imposed directly on the views (global relations)

No guarantee for their satisfaction (data stay at the sources)



5. Updates through Views

Another related classic problem in relational DBs

Given:

- Base schema $\mathcal{S}$
- View definition: $\forall \bar{x}(V(\bar{x}) \leftrightarrow \varphi_{\mathcal{S}}(\bar{x}))$ (*)
- An instance D for $\mathcal{S}$, and extension $V(D)$ for the view

Apply update U on $V(D)$, propagating updates on D, keeping (*) satisfied

Example: $V(x, y) \leftrightarrow Manager(x, y, z) \wedge z = 100$

Manager	Boss	Subordinate	Salary
	ken	$john$	100
	$john$	$mary$	120
	$peter$	joe	150

Easy!

Less easy with base ICs, e.g. $Manager : Boss, Subordinate \rightarrow Salary$

However, for more complex queries ...

Example: $D = \{R(a,b), R(c,d), S(b,c)\}$

$$V_2(x,y) \leftarrow R(x,z), S(z,y)$$

U: $insert_{V_2}(a,d)$

$D' = \{R(a,b), R(c,d), S(b,c), R(a,NULL), R(NULL,d)\}$?

Ordinary SQL Nulls? (in joins?)

Arbitrary values from the domain?

Conditional instances?

$D' = \{R(a,b), R(c,d), S(b,c), R(a,X), R(X,d)\}$

What if also base IC $R : A \rightarrow B$?

Disjunctive views?

$V_3(x, y) \leftarrow R(x, y)$
$V_3(x, y) \leftarrow S(x, y)$

$U: \ insert_{V_3}(e, d)$

$D' = \{R(a, b), R(c, d), S(b, c), R(e, d)\}$?

$D' = \{R(a, b), R(c, d), S(b, c), S(e, d)\}$?

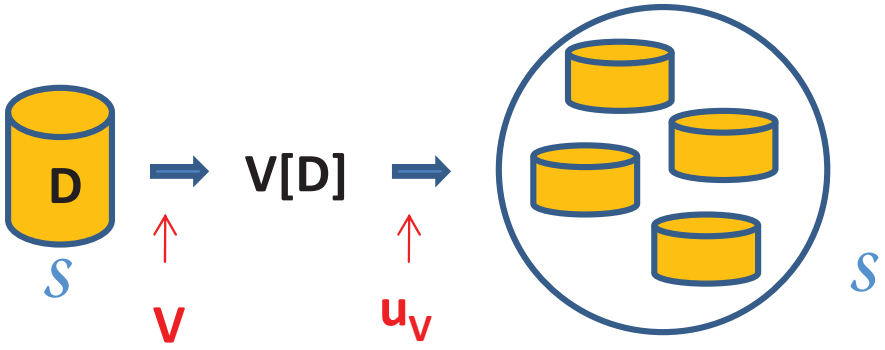
$D' = \{R(a, b), R(c, d), S(b, c), R(e, d), S(e, d)\}$?

Several choices ... Which are the right ones?

It depends on the update semantics (of DBs through views)

It can be a "possible world semantics"

A class of intended (admissible, legal) instances D' that reproduce the view update U



Preference criteria can be imposed on elements of the class

An old and difficult problem in relational databases

Semantics and algorithms based on assumption of availability of a view complement [Bancil. et al. 81; Cosm. et al. 84; ...; Lecht. et al. 03]

No standard solution offered/implemented in commercial DBMS

In practice:

- Some views are considered to be updatable

  Restrictions on views make them updatable or not

- `INSTEAD-OF` triggers

  Instead of direct view update, update base tables (as indicated by trigger), causing the intended change on the view

Different approaches to update through views:

- Abductive: Relationship between views V and base tables T given by view definitions plus base ICs

Observations are the intended view updates ($^{+}_{-}atom_V$)

Abductibles are $^{+}_{-}atom_T$, those that explain (cause) the observations

Abductive logic programming, including ICs [Kakas et al. 90, 92]

- ASPs: Possible worlds as stable models of a disjunctive ASP specifying how view and base updates are related [LB et al. 11]

Remark: In virtual data integration, the corresponding problem would be updating the sources through the mediator (containing views under GAV) Not allowed in general, but [De Giac. et al. 09]

6. Inconsistency Handling

What If the database is inconsistent?

Inconsistencies can be detected, and data can be changed to reach a physical consistent state

This kind of data cleaning may be difficult, impossible, nondeterministic, undesirable, unmaintainable, etc.

We may have to live with inconsistent data ...

The database (the model) is departing from the outside reality that is being modeled

However, the information is not all semantically incorrect

Most likely most of the data in the database are still "consistent"

Idea:

(a) Keep the database as it is

(b) Obtain semantically meaningful information at query time; dealing with inconsistencies on-the-fly

Particularly appealing in virtual data integration ...
(no direct access to the data sources)

This requires: [LB 11]

(a) Logically characterizing consistent data within an inconsistent database

Via database repairs: Consistent instances that minimally depart from the original instance

Consistent data is invariant across the class of all repairs

(b) Developing algorithms for retrieving the consistent data: Consistent query answering

Example: For the instance D that violates FD: $Name \rightarrow Salary$

Employee	*Name*	*Salary*
	page	5K
	page	8K
	smith	3K
	stowe	7K

Two possible (minimal) repairs if only deletions/insertions of whole tuples are allowed: D_1, resp. D_2

Employee	*Name*	*Salary*
	page	5K
	smith	3K
	stowe	7K

Employee	*Name*	*Salary*
	page	8K
	smith	3K
	stowe	7K

$(stowe, 7K)$ persists in all repairs: it is consistent information

$(page, 8K)$ does not; actually it participates in the violation of FD

A consistent answer to a query Q from a database D is an answer that can be obtained as a usual answer to Q from every possible repair of D wrt IC (a given set of ICs)

- $Q_1 : Employee(x, y)?$

  Consistent answers: $(smith, 3K), (stowe, 7K)$

- $Q_2 : \exists y Employee(x, y)?$

  Consistent answers: $(page), (smith), (stowe)$

CQA may be different from classical data cleaning!

However, CQA is relevant for data quality; an increasing need in business intelligence

It also provides concepts and techniques for data cleaning

Paradigm shift: ICs are constraints on query answers, not on database states!

Depending on the ICs and the queries, tractable and intractable cases for CQA have been identified

For some tractable cases, query rewriting algorithms have been developed

$$\mathcal{Q}(x, y): \ \mathit{Employee}(x, y) \quad \mapsto$$
$$\mathcal{Q}'(x, y): \ \mathit{Employee}(x, y) \wedge \neg \exists z(\mathit{Employee}(x, z) \wedge z \neq y)$$

Pose the second query to the inconsistent database to obtain the consistent answers to the first query

For higher-complexity cases, specifications of repairs by means of logic programs with stable model semantics can be used

CQA becomes querying (as usual) a logic program, say a Datalog program with possible complex extensions

7. Specifying DB Evolution

So far here, we have a logical specification of the DB, but external updates that change the DB

We can integrate everything into a single logical theory that specifies the DB and its evolution

For that we need the right language

Situation Calculus

A family of languages of many-sorted first-order logic

Used in logic-based KR to describe evolving domains subject to the execution of actions

Regained popularity in the 90's due mainly to the work of Raymond Reiter and collaborators [Reiter 01]

- A simple solution to the *frame problem* in the SC

  Given a specification of precondition and effects of actions, how to obtain a compact, economical specification of the many things that are not changed by the actions

- Basis for cognitive robotics programs: GOLOG, CONGOLOG

The Languages

- Domain individual, situations (states), and actions at first-order level

- First–order quantifications over sorts: $\forall \bar{x}$, $\forall s$, $\forall a$

- S_0, name for initial situation

- Function name do: $do(a, s)$ is the successor state that results from executing action a at state s

- Predicate $Poss$: $Poss(a, s)$ says that action a is possible at state s

- Parameterized action terms, e.g. $promote(x, p)$

- Predicates with situation argument, e.g. $Enrolled(x, p, s)$

- Static predicates

Foundational Axioms for the SC

Unique Names Axioms for Actions: $a_i(\bar{x}) \neq a_j(\bar{y})$, for all different action names a_i, a_j, e.g. $delete(id) \neq classifyBook(isbn, id')$

Unique Names Axioms for States: $S_0 \neq do(a, s)$

$$do(a_1, s_1) = do(a_2, s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2$$

For some reasoning tasks the *Induction Axiom on States* (IA):

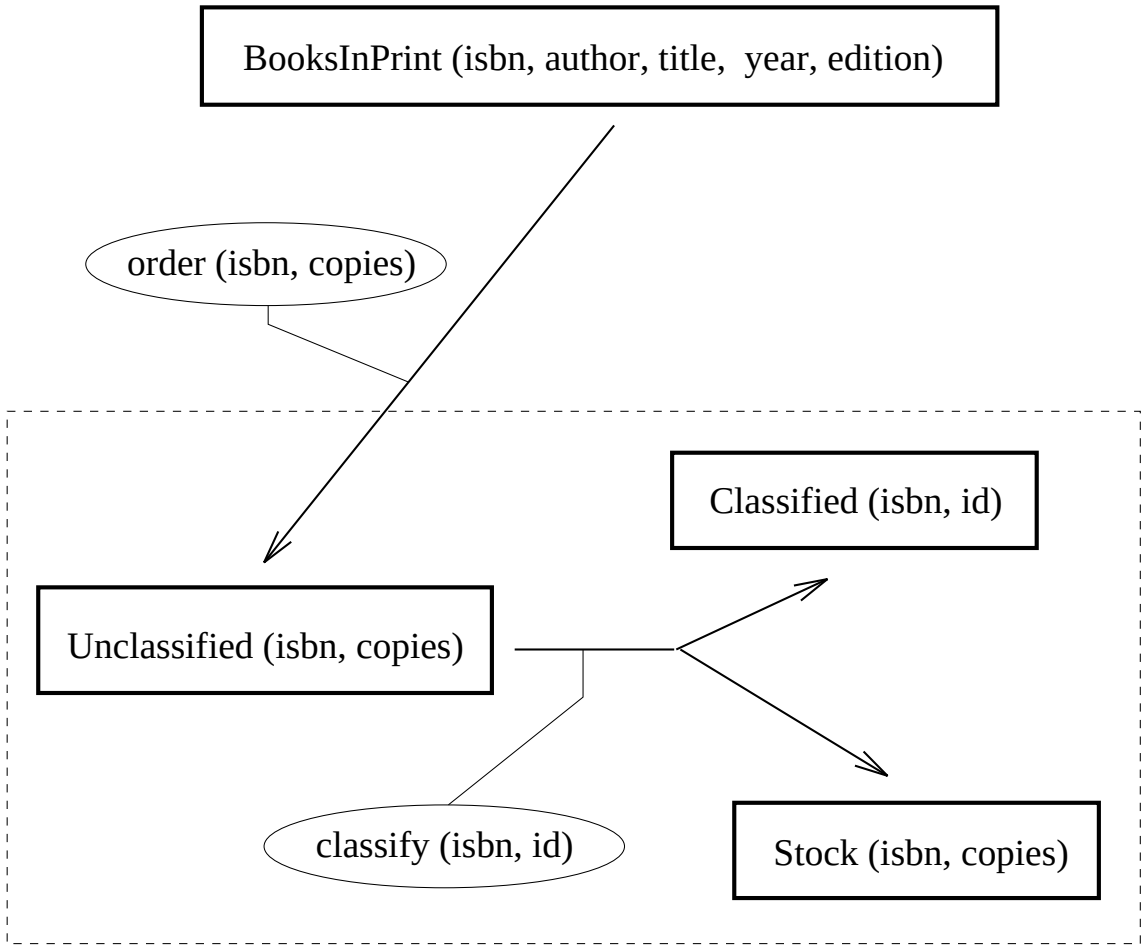
$$\forall P\ (P(S_0) \wedge \forall s \forall a\ (P(s) \rightarrow P(do(a, s))) \rightarrow \forall s\ P(s))$$

restricts the domain of situations to S_0 plus the situations obtained by executing actions

We are usually interested in reasoning about states that are *accessible* from S_0 by executing a finite sequence of legal actions:

$$\neg s < S_0 \qquad\qquad s < do(a, s') \equiv Poss(a, s') \wedge s \leq s'$$

Specifying DB Updates



Update actions: $order(isbn, copies)$, $classifyBook(isbn, id)$, $deleteBook(id)$

Predicates: $BooksInPrint(isbn, title, author, editor, year, edition)$, $Unclassified(isbn, copies, s)$, $Classified(isbn, id, s)$, $Stock(isbn, copies, s)$

Action Preconditions:

- $Poss(order(isbn, copies), s) \equiv$
  $(\exists\ title, author, editor, year, edition)$
  $BooksInPrint(isbn, title, author, editor, year, edition)$

- $Poss(classifyBook(isbn, id), s) \equiv$
  $\neg(\exists\ isbn')\ Classified(isbn', id, s) \wedge$
  $(\exists\ copies)\ Unclassified(isbn, copies, s)$

Successor State Axioms

Solution to the frame problem is based on the use (generation) of successor state axioms (Reiter 91)

Specify under what conditions each fluent becomes true at an arbitrary successor state $do(a, s)$

$$\forall a \forall s \; Poss(a, s) \; \rightarrow \; [R(do(a, s)) \; \equiv \; \gamma_R^+(a, s) \vee (R(s) \wedge \neg \gamma_R^-(a, s))]$$

R is true at a successor state iff it is made true or it was already true and it is not made false

This solution relies on the possibility of quantifying over deterministic actions

Actions are syntactically atomic, but semantically complex

In DB applications (Reiter 95), individual actions may result in several DB updates

A SSA

$$
\begin{aligned}
&\forall a\ Poss(a,s) \longrightarrow \\
&\quad (Stock(isbn,j,do(a,s)) \equiv \\
&\qquad a = delete\_book(id) \wedge \\
&\qquad\quad (Classified(isbn,id,s) \wedge \\
&\qquad\qquad \exists i\ (Stock(isbn,i,s) \wedge i > 1 \wedge j = i-1)) \vee \\
&\qquad a = classify\_book(isbn,id) \wedge \\
&\qquad\quad (\exists i\ (Stock(isbn,i,s) \wedge j = i+1) \vee \\
&\qquad\qquad \neg\exists i\ (Stock(isbn,i,s)) \wedge \\
&\qquad\qquad\quad j = 1) \vee \\
&\qquad Stock(isbn,j,s) \wedge \\
&\qquad\quad \neg(a = delete\_book(id) \wedge \\
&\qquad\qquad\quad Classified(isbn,id,s) \wedge \\
&\qquad\qquad\qquad Stock(isbn,j,s) \vee \\
&\qquad\qquad a = classify\_book(isbn,id) \wedge \\
&\qquad\qquad\quad Stock(isbn,j,s)))
\end{aligned}
$$

(can be constructed from positive and negative effect axioms)

Integrity Constraints

Static ICs, e.g. FDs, are sentences that must hold at every legal state of the database:

$$DB\ spec. \models \forall s\ (S_0 \leq s \rightarrow \varphi(s))$$

$$\uparrow$$

$$Th(DB(S_0)) \cup\ Spec.\ of\ Dynamics$$

For example, for a FD it should hold

$$DB\ spec. \models \forall S_0 \leq s \rightarrow (Classified(isbn1, id, s) \wedge Classified(isbn2, id, s) \rightarrow isbn1 = isbn2)$$

Induction principle for proving static ICs can be derived:

$$\forall P([P(S_0) \wedge \forall s \forall a(P(s) \wedge Poss(a, s) \rightarrow P(do(a, s))] \rightarrow \forall s(S_0 \leq s \rightarrow P(s)))$$

Similar treatment for dynamic ICs:

A person's salary cannot decrease:

$$\forall s, s'(S_0 \leq s \leq s' \ \rightarrow \ (Salary(x, p, s) \wedge Salary(x, p', s') \ \rightarrow \ p \leq p'))$$

ICs proved by automated mathematical induction [LB et al. 96]

Specifying the Dynamics of Relational Views

Given a specification of DB dynamics in terms of SSAs

Automatically derive SSAs for (relational calculus) views

Applications to view and database maintenance [Arenas et al. 98]

Can be extended to aggregate views

Combination with hypothetical reasoning? "What if" queries?

Hypothetical Database Reasoning [Arenas et al. 02]

Queries in first-order past temporal logic about a whole evolution of the database

Application to transformation of dynamic ICs into static ICs

Application to transformation of history dependent actions into "Markovian" actions

Change and Ontologies

Complex actions can be constructed from basic actions (cf. GOLOG)

It is possible to derive SSAs from complex actions [Fritz et al. 08]

They and GOLOG used for composition of semantic web services [McIlr. et al. 02]

Actually, generic ontologies (ontology languages) have been proposed for specifying action and change

General ontologies $\mathcal{O}^g$ for high-level descriptions of action preconditions, actions effects, etc. (e.g. OWL-S, FLOWS) [Martin et al. 07, Grün. et al. 08]

Specific theories $\mathcal{O}^s$ of action and change (as in the library example above) can feed $\mathcal{O}^g$

The combination can be applied to specify the evolution of an initially static domain, e.g. a database D

A three-layered approach ...

References:

[Arenas et al. 98] Arenas, M., Bertossi, L. The Dynamics of Database Views. In 'Transactions and Change in Logic Databases', Springer LNCS 1472, (1998)

[Arenas et al. 02] Arenas, M., Bertossi, L. Hypothetical Temporal Queries in Databases. Journal of Intelligent Information Systems, 2002, Volume 19, Number 2, pp. 231-259.

[Bancil. et al. 81] Francois Bancilhon, Nicolas Spyratos: Update Semantics of Relational Views. ACM Trans. Database Syst. 6(4): 557-575 (1981)

[LB et al. 96] Bertossi, L., Pinto, J., Saez, P., Kapur, D., Subramaniam, S. Automating Proofs of Integrity Constraints in the Situation Calculus. In 'Foundations of Intelligent Systems', Proc. ISMIS'96, Springer LNAI 1079 (1996)

[LB et al. 11] Leopoldo Bertossi, Lechen Li: Achieving Data Privacy through Secrecy Views and Null-Based Virtual Updates CoRR abs/1105.1364: (2011). To appear in IEEE TKDE.

[LB 11] Leopoldo Bertossi: Database Repairing and Consistent Query Answering. Morgan & Claypool Publishers (2011)

[Blakeley et al. 89] Jose Blakeley, Neil Coburn, Per-Ake Larson: Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. ACM Trans. Database Syst. 14(3): 369-400 (1989)

[Cosm. et al. 84] Stavros S. Cosmadakis, Christos H. Papadimitriou: Updates of Relational Views. J. ACM 31(4): 742-760 (1984)

[De Giac. et al. 09] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati: On Instance-level Update and Erasure in Description Logic Ontologies. J. Log. Comput. 19(5): 745-770 (2009)

[Fritz et al. 08] Fritz, C.; Baier, J. A.; and McIlraith, S. A. ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for Planning and Beyond. Proceedings KR'08 (2008)

[Godfrey et al. 98] Parke Godfrey, John Grant, Jarek Gryz, Jack Minker: Integrity Constraints: Semantics and Applications. Logics for Databases and Information Systems 1998: 265-306

[Greco et al. 12] Sergio Greco, Cristian Molinaro, Francesca Spezzano: Incomplete Data and Data Dependencies in Relational Databases. Morgan & Claypool Publishers (2012)

[Grün. et al. 08] Michael Grüninger, Richard Hull, Sheila A. McIlraith. Bull. IEEE CS on Data Engineering (2008)

[Gupta et al. 95] Ashish Gupta, Inderpal Singh Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. IEEE Data Eng. Bull. 18(2): 3-18 (1995)

[Gupta et al. 99] Ashish Gupta, Inderpal Singh Mumick (eds.): Materialized Views. Morgan Kaufmann (1999)

[Kakas et al. 90] Antonis C. Kakas, Paolo Mancarella: Database Updates through Abduction. VLDB 1990: 650-661

[Kakas et al. 92] Antonis C. Kakas, Robert A. Kowalski, Francesca Toni: Abductive Logic Programming. J. Log. Comput. 2(6): 719-770 (1992)

[Klug 80] Anthony C. Klug: Calculating Constraints on Relational Expressions. ACM Trans. Database Syst. 5(3): 260-290 (1980)

[Klug 82] Anthony C. Klug, Rod Price: Determining View Dependencies Using Tableaux. ACM Trans. Database Syst. 7(3): 361-380 (1982)

[Lecht. et al. 03] Jens Lechtenboerger, Gottfried Vossen: On the computation of relational view complements. ACM Trans. Database Syst. 28(2): 175-208 (2003)

[Martin eta al. 07] David Martin, Mark Burstein, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Katia Sycara, Deborah L. McGuinness, Evren Sirin, Naveen Srinivasan: Bringing Semantics to Web Services with OWL-S. World Wide Web 10:243-277 (2007)

[McIlr. et al. 02] McIlraith, S., and Son, T. Adapting Golog for Composition of Semantic Web Services. 2002. Proc. KR'02 (2002)

[Reiter 84] Raymond Reiter: Towards a Logical Reconstruction of Relational Database Theory. In "On Conceptual Modeling", M. Brodie, J. Myolopoulos, J. Schmidt (eds.), Springer (1984)

[Reiter 92] Raymond Reiter: What Should a Database Know? J. Log. Program. 14(1&2): 127-153 (1992)

[Reiter 01] Raymond Reiter: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press (2001)

[Widom et al. 95] J. Widom, S. Ceri. Active Database Systems: Triggers and

Rules for Advanced Database Processing. Morgan Kaufmann (1995)