Managing Virtual Records in Relational Databases

WATERLOO
CHERITON SCHOOL OF COMPUTER SCIENCE
cs.uwaterloo.ca

Frank Wm. Tompa
Ahmed Ataullah

ACKNOWLEDGEMENTS

"Records Retention in Database Management Systems," in *CIKM 2008*, 873-882 (co-authored with Ashraf Aboulnaga).

"Records Retention in Relational Database Systems: Bridging the Gap between Laws and Enforcement Actions," in *RELAW 2009*: 15-16.

"Lifecycle Management of Relational Records for External Auditing and Regulatory Compliance," in *POLICY 2011*: 73-80.

"Business Policy Modeling and Enforcement in Databases," *PVLDB 4*(11): 921-931 (2011).

Research funded in part by NSERC BIN Strategic Network, RIM, and Open Text Corporation.

RECORDS MANAGEMENT (RM)

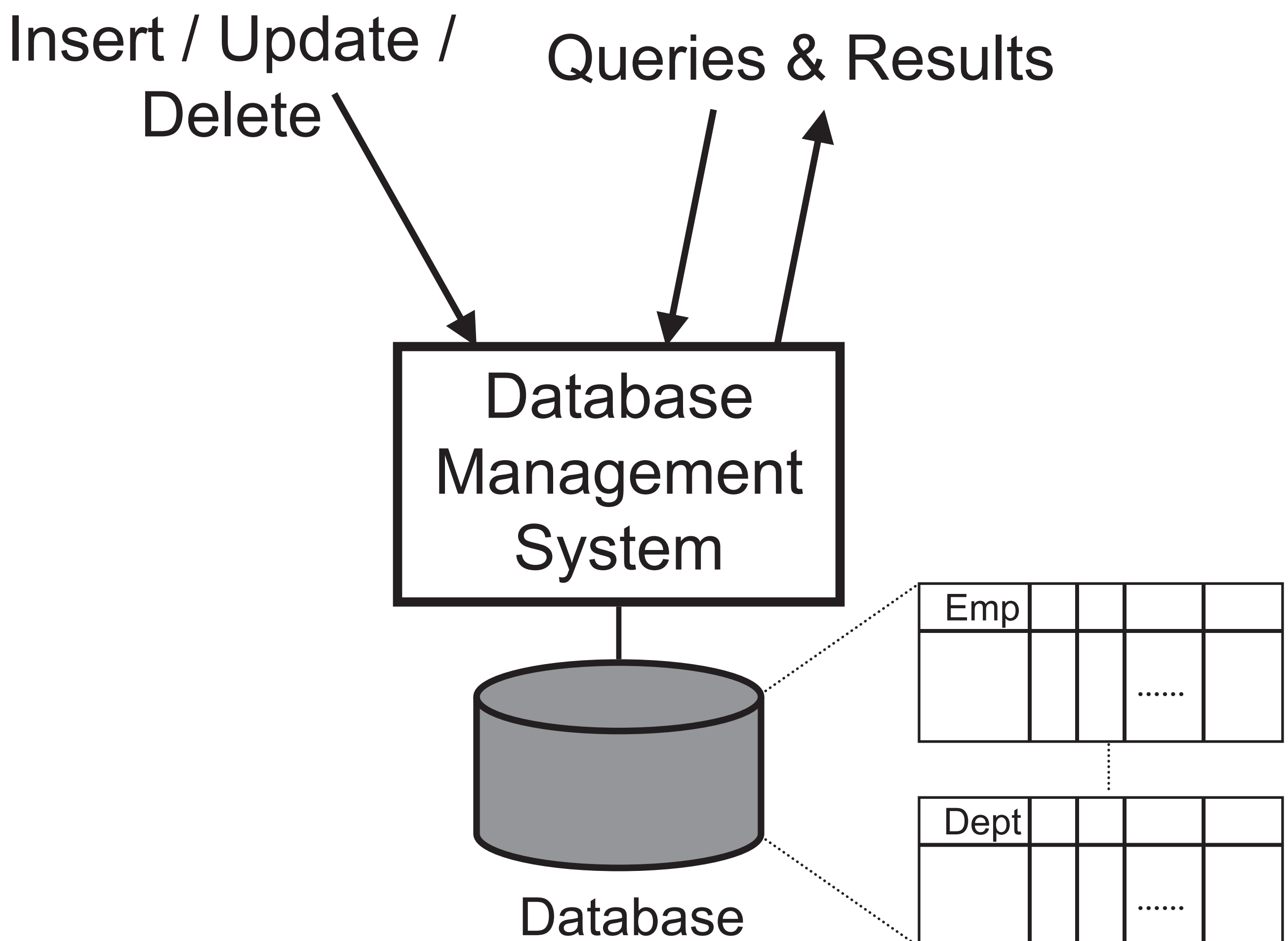
Enterprises must ensure business records (typically documents) are available for

- Operational needs
- Legal / regulatory compliance
- Fiscal management
- Other vital information
- Historical value

Key aspects:

- Preservation
- Accuracy
- Efficient access

WHY DATABASES?

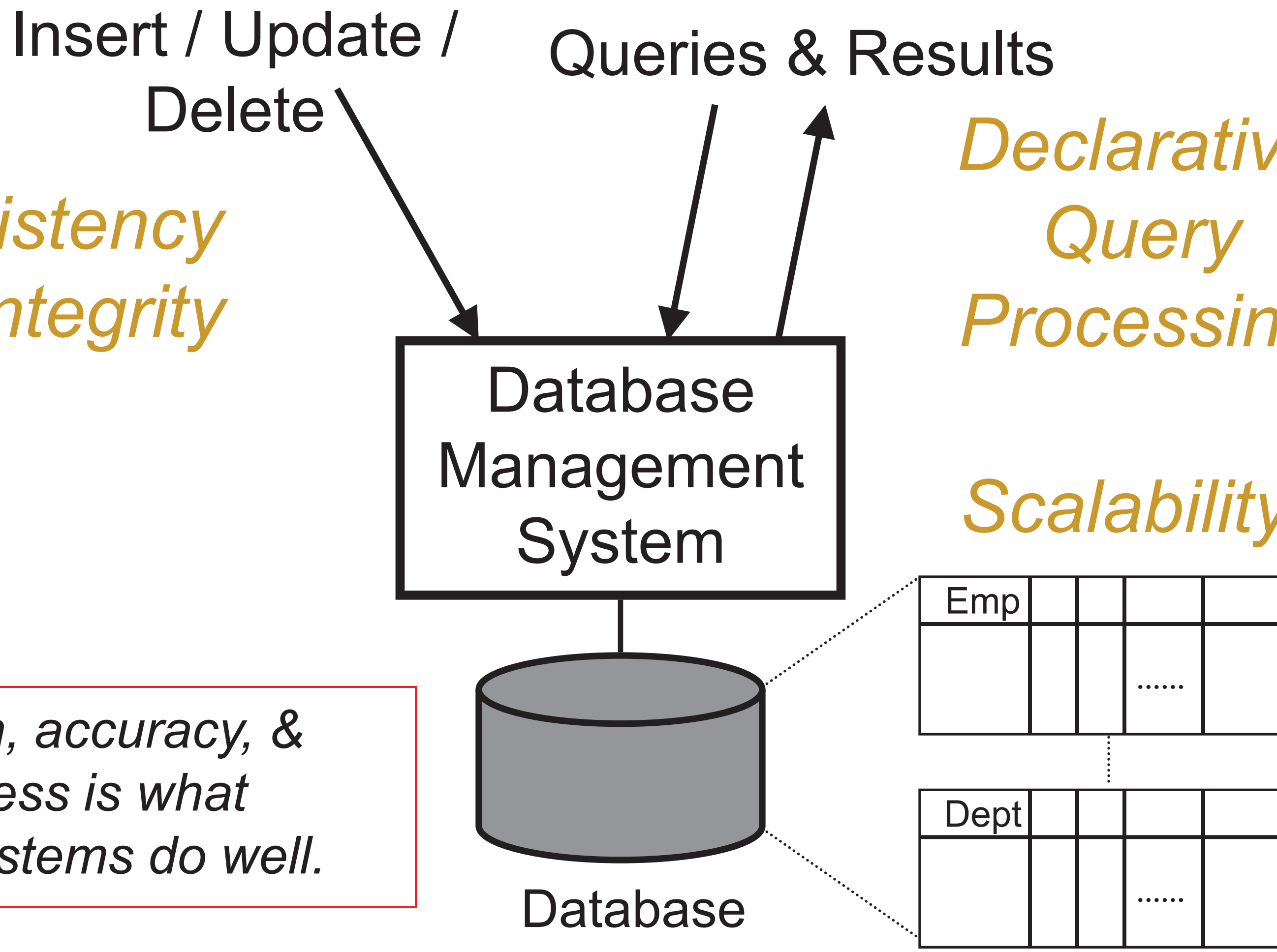


WHY DATABASES?

Consistency and Integrity

Declarative Query Processing

Scalability



Preservation, accuracy, & efficient access is what database systems do well.

RM INCLUDES DESTRUCTION

*Absent extraordinary circumstances, if an organization has implemented a clearly defined records management program specifying what information and records should be kept for legal, financial, operational or knowledge value reasons and has set appropriate retention systems or periods, then information not meeting these retention guidelines can, and should, be destroye*d. [Sedona Guidelines]

Improved efficiency (space and time)

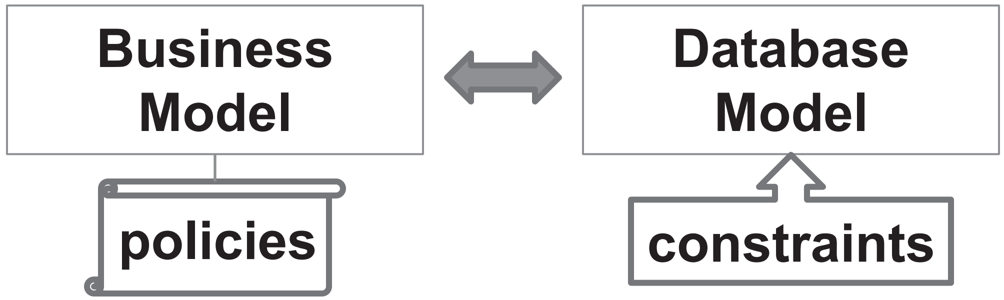
- *More applicable to physical records*

Protection during litigation or government investigation

- Avoid superfluous liability
- Avoid releasing collateral information; protect privacy
- Must prove it was destroyed as part of corporate *retention policy* (documented and consistently applied) and it is *irrecoverable*

Database systems are *not* good at this.

RM AS BUSINESS POLICY



Business policy:

- Summary of a set of rules; a high-level overview
- Specification of what should (or should not) happen in the operations of a business
  - Typically written in natural language
  - Policy modeling: Hierarchical access control, Object Constraint Language, …
- Includes records management, workflow, privacy, regulatory compliance, etc.

Business policy model implemented by DB policy model

DATABASE SYSTEM PERSPECTIVE

DB instance must comply with published policy

- DBMS continuously monitors updates to ensure compliance.
  - Using check constraints and transaction termination triggers
  - Access control restrictions
- DBMS provides a single compliance layer:
  - No need for policy checking logic in every application
  - Platform for detecting policy conflicts and guaranteeing compliance

Need policy-to-constraint clarity and manageability

- How to make the task of the programmer easier?
  - Mapping business policies $\rightarrow$ DB constraints
- How to make database rules understandable to business managers?
  - Mapping business policies $\leftarrow$ DB constraints

DIFFICULTIES

Each department deals with separate policies.

Many constraints are complex (temporal, conditional, path oriented).

- Typically correspond to complex triggers invoked by transaction termination
- e.g., *A private physician in Ontario must keep patient records for 15 years after the last entry in the record, or for 10 years after the patient turns 18 years old, whichever comes later, but cannot delete a record containing information that has been requested under PHIPA nor a record subject to a litigation hold.*

Business policies across enterprise produce many constraints

- Scale ⇒ manageability a major problem for DB administrators and programmers

TALK OUTLINE

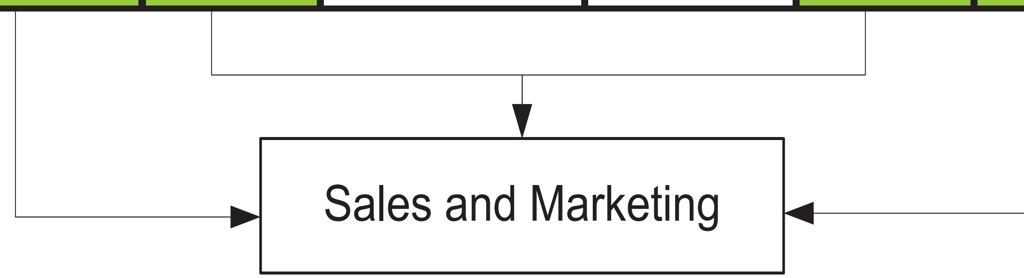
- ➢ **What is a record in a relational database?**
- ➢ **How can record lifecycles be specified?**
- ➢ **How can update constraints be enforced?**
- ➢ **How can record disposition be enforced?**

Disclaimer

- Mechanisms described are not available today as tools
- Mechanisms can all be implemented with today's tools
  - Relational database management systems
  - Symbolic model checkers

RECORDS FROM DATABASES

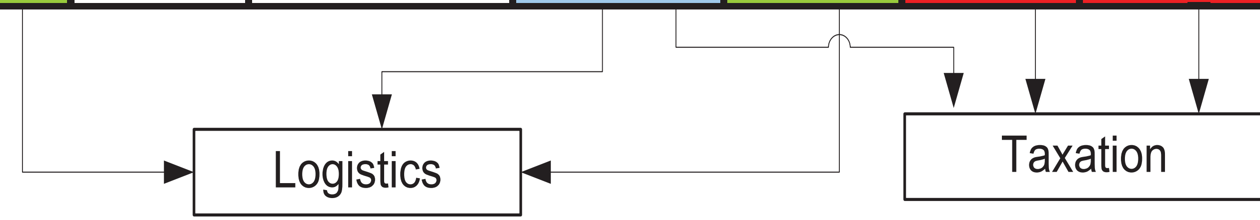
Invoice	LineItem	Product	Supplier	CommitDate	ShipRef	Quantity	Amount	TaxCode
345	1	55	44	1/1/2005	534	1	34.95	US_NY
345	2	111	22	17/01/2005	null	50	79.95	null
…	…	…	…	…	…	…	…	…
399	1	23	62	24/4/2006	1234	3	29.95	VAT_GB
450	1	44	34	24/4/2006	2243	1	39.95	US_IL



A record is data presented in a certain context such that it holds meaning for a user.

OVERLAPPING RECORDS

Invoice	LineItem	Product	Supplier	CommitDate	ShipRef	Quantity	Amount	TaxCode
345	1	55	44	1/1/2005	534	1	34.95	US_NY
345	2	111	22	17/01/2005	null	50	79.95	null
…	…	…	…	…	…	…	…	…
399	1	23	62	24/4/2006	1234	3	29.95	VAT_GB
450	1	44	34	24/4/2006	2243	1	39.95	US_IL



Underlying data may be shared

- Multiple policies can apply to a single datum
- Retention policies must be enforced carefully

DB BUSINESS RECORD

Formally, a *record type* is a logical view specified over a fixed physical schema

- The result of any (meaningful) query posed to a database
- Class of records
- *Record* is a uniquely identifiable row in such a view

Specified by

- Policy managers, DBA, data analysts, lawyers

Record as row in a *view* ⇒

- More expressive than rows in a base table
- Unambiguous, declarative, subject to formal reasoning
- Many already defined for operational reasons
- Represents the states of a business object

RECORD STATES

State = condition on records in the view

- Object x is in state S ⇔ its attributes satisfy S(x)
- Example

```
define record UPS_ORDERS as
select *
from (ORDERS natural join CUSTOMERS) left outer join
        SHIPMENTS on ORDERS.ShipRef = SHIPMENTS.ShipRef
where ShipMethod = 'UPS'
```

  - UPS_ORDERS is user-defined view
  - “Order objects” are rows in the view

```
define state Paid on UPS_ORDERS
where Paid = true
```

  - Object O (tuple t) is in state P, the “Paid state”, if the condition UPS_ORDERS.Paid = true for t

🗝 **An object can be in multiple states at once.**

TYPICAL WORKFLOW

In workflows, for example,

- Rectangles represent business states
- Transitions represent processes/actions
- Stick-figures represent agents
- Constraints implied by absence of transitions
    - E.g., Paid orders cannot go back to awaiting approval

BUSINESS STATES AND DB STATES

States of an object are typically an interpretation of stages in business process.

BUSINESS STATES AND DB STATES

States of an object are typically an interpretation of stages in business process.

- Homomorphism from business states to DB states

BUSINESS STATES AND DB STATES

Awaiting Approval
Approved = false

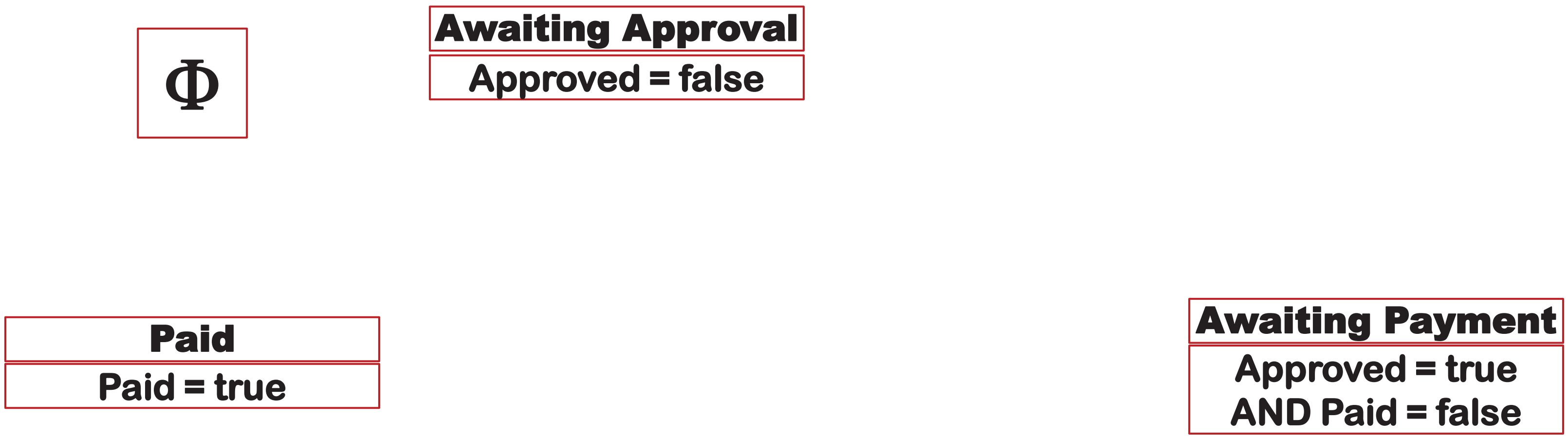
Paid
Paid = true

Awaiting Payment
Approved = true
AND Paid = false

States of an object are typically an interpretation of stages in business process.

- Homomorphism from business states to DB states

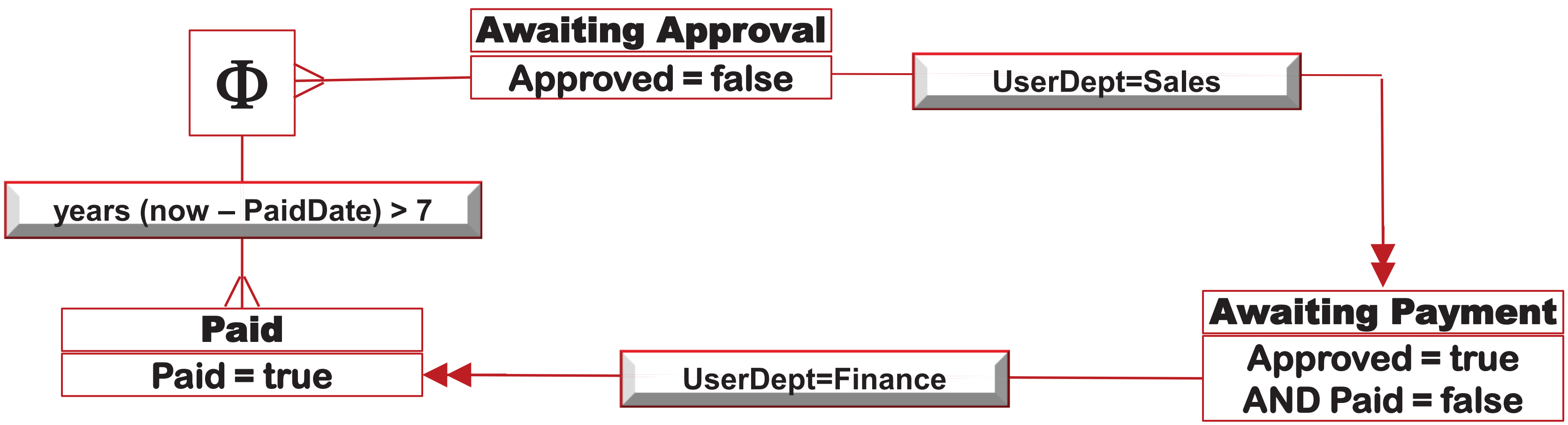
BUSINESS STATES AND DB STATES



States of an object are typically an interpretation of stages in business process.

- Homomorphism from business states to DB states
- Including object creation and destruction

BUSINESS STATES AND DB STATES

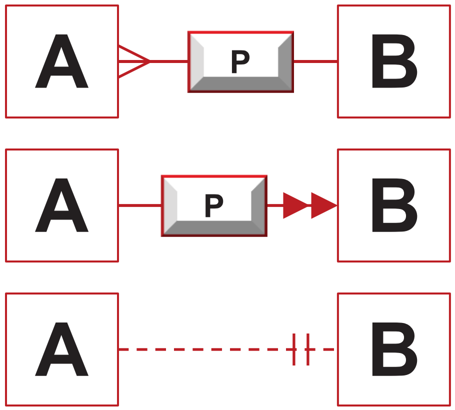


States of an object are typically an interpretation of stages in business process.

- Homomorphism from business states to DB states
- Including object creation and destruction

All constraints should be made explicit in such a constraint diagram.

CONSTRAINTS ON TRANSITIONS



$$\bullet A(x) \wedge \neg A(x) \Rightarrow B(x) \wedge P$$

$$\neg \bullet B(x) \wedge B(x) \Rightarrow \bullet A(x) \wedge P$$

$$\blacklozenge A(x) \Rightarrow \neg B(x)$$

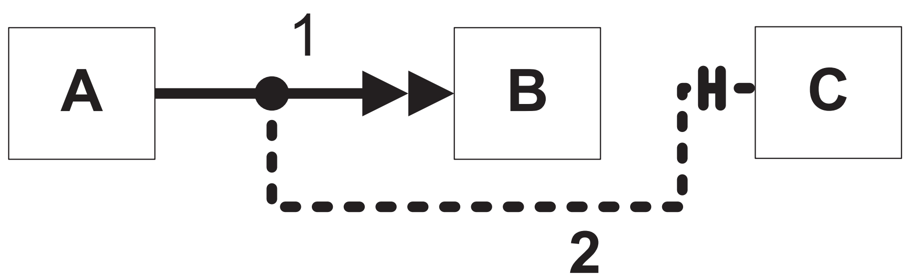
Define various types of constraint

- Syntax: diagrammatic form
- Sematics: using past temporal logic

Special interpretations for transitions to or from Φ

ALLOW COMPLEX PATH CONSTRAINTS

For example, an object should never reach state C if it has previously transitioned from A to B



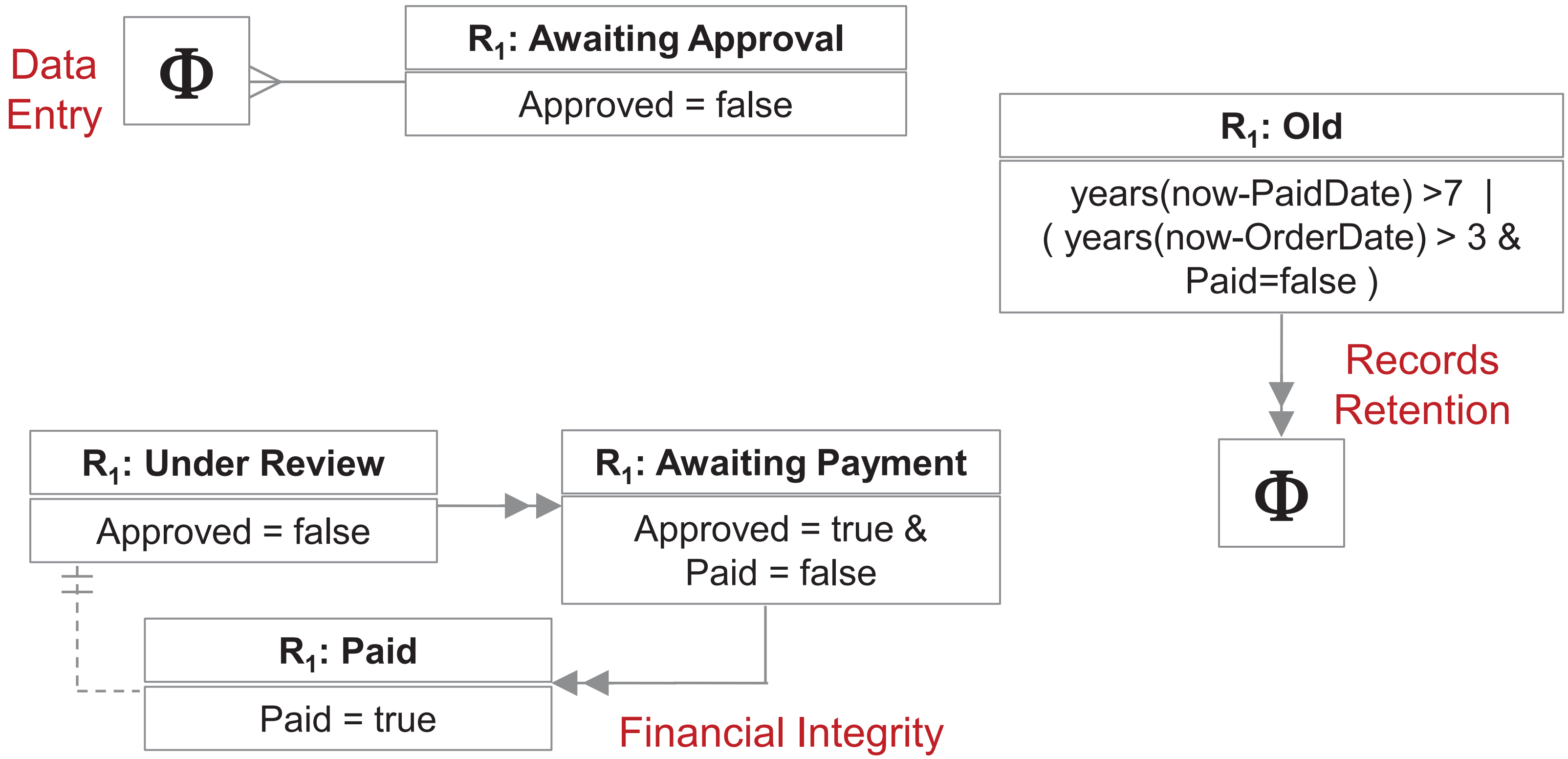
Constraint 1 : B(r) ∧ ¬●B(r) ⇒●A(r)

Constraint 2 : ♦1(r) ⇒ ¬C(r)

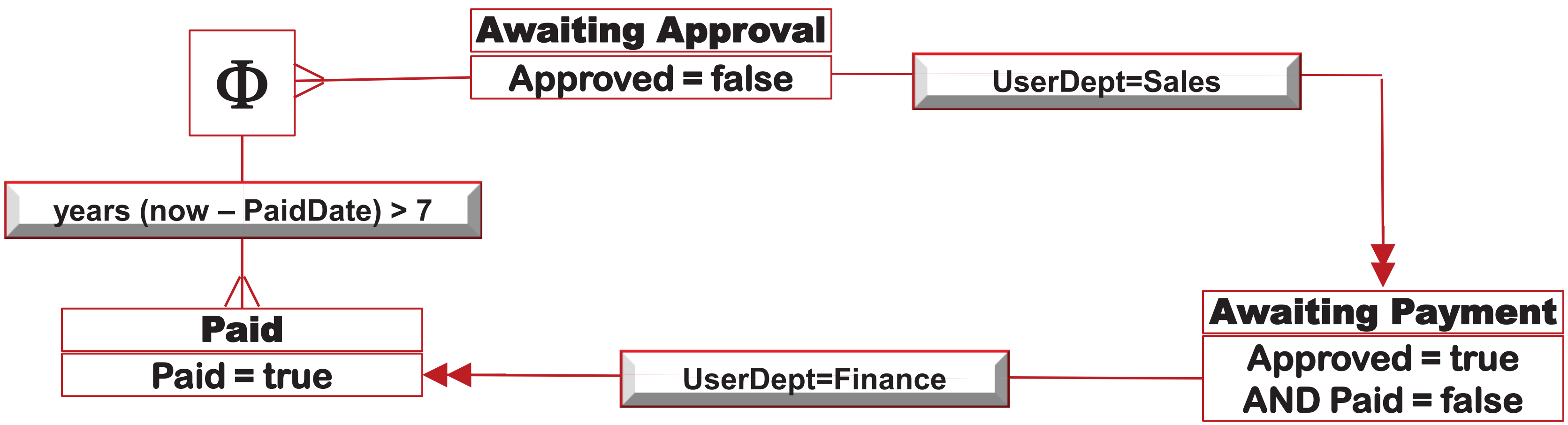
Resulting constraint : ♦(B(r) ∧ ¬●B(r)) ⇒ ¬C(r)

ALLOW MULTIPLE DIAGRAMS

Model divisions' business processes as individual constraint diagrams



BACK TO THE EXAMPLE



Convert to temporal logic

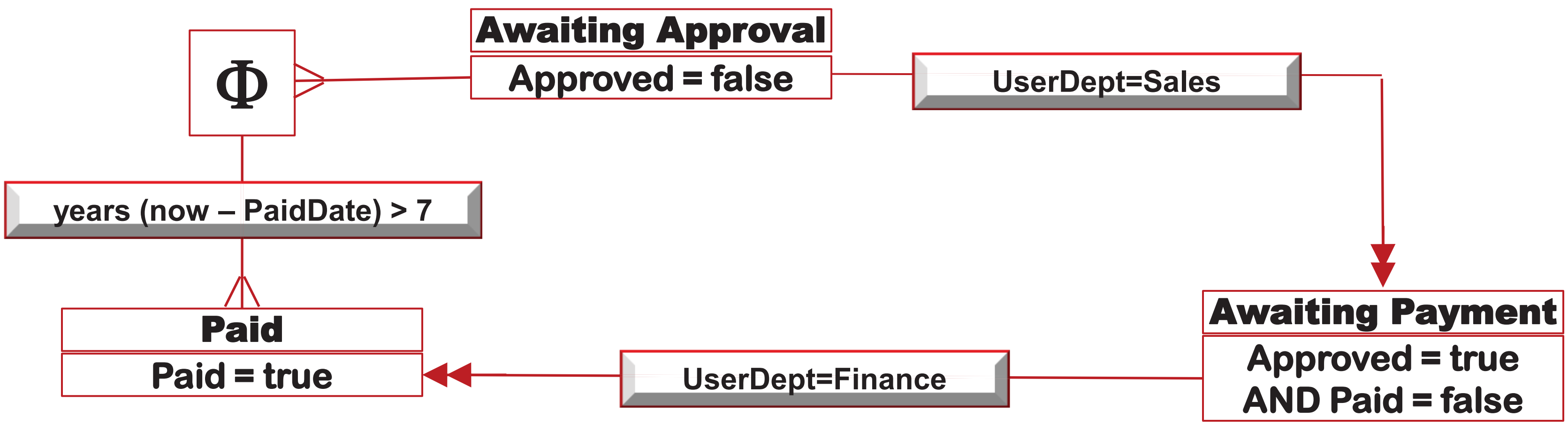
$New(x) \Rightarrow \text{AwaitingApp}(x)$

$\neg \bullet\text{AwaitingPay}(x) \land \text{AwaitingPay}(x) \Rightarrow \bullet\,\text{AwaitingApp}(x) \land \text{UserDept} = \text{Sales}$

$\neg \bullet\text{Paid}(x) \land \text{Paid}(x) \Rightarrow \bullet\,\text{AwaitingPay}(x) \land \text{UserDept} = \text{Finance}$

$\text{Paid}(x)\ \&\ \text{years}(\text{now-PaidDate}(x)) \leq 7 \Rightarrow Retain(x)$

STATES OF AN OBJECT



States = { AwaitingApproval,
AwaitingPayment,
Paid}

State space = {(0,0,0),(0,0,1),(0,1,0),…,(1,1,1)}

- Some configurations are not satisfiable
- Others may be subject to policy constraints

Property verification: use model checker to prove consistency, etc.

STATE CONFIGURATION HISTORY

Complete, *temporally ordered* list of state configurations per object

Object O_1 history

Time	Awaiting Approval	Awaiting Payment	Paid
t1	1	0	0
t2	0	1	0
t3	0	1	0
…	…	…	…
t_new	*1*	*0*	*1*

Enforcement: every time an object is modified, look back at history and check each constraint

- Experiments show that constraint checking overhead is insignificant.

RECORDS DISPOSITION POLICIES

Objective: to destroy (or anonymize) records or to archive them once certain conditions are met

- Condition typically triggered by advance of time
- Requires database to take action
- Cf. Other policies *disallow* classes of updates

Disposition states similar to other DB states

- *But* records in disposition states *must* be archived, deleted, or altered

Disposition transitions can be scheduled

- Cf. Other policy transitions must be continuously enforced
- When to invoke transition balances efficiency vs. risk
- Is each disposition transition effective?
- Do combined disposition transitions always terminate?

CONCLUSIONS

Emphasizes data, but models processes and user interactions

Policy designer only needs to list the “states of interest”

- By specifying the conditions that the object must satisfy to be in those states
- Each policy designer in the organization can list his/her own states of interest

Policy restricts paths that objects can (or should) traverse

- Constraint diagram (the model)
  - Some paths must always be taken, some must never be traversed, and others can be conditionally traversed

Database system enforces constraints automatically

- Every update transaction triggers check of objects’ state configuration history
- Disposition transitions checked and enforced as per schedule

READ MORE

- Atalluah & Tompa: *Proc. CIKM'08, RELAW'09, POLICY'11, PLVDB'11*
- Jacoby (ed.): *The Sedona Conference® Database Principles, Addressing the Preservation & Production of Databases & Database Information in Civil Litigation*, March 2011 Public Comment Version, 47 pp.
- Wagner (ed.): *The Sedona Guidelines: Best Practice Guidelines & Commentary for Managing Information & Records in the Electronic Age* (2nd Edition), November 2007, 91 pp.
- Hasan, Sion, & Winslett: "Preventing history forgery with secure provenance," *TOS 5*(4) (2009).
- Mitra, Winslett, Snodgrass, Yaduvanshi, & Ambokar: "An Architecture for Regulatory Compliant Database Management," in *ICDE 2009*: 162-173.

Thank you