A Mini-course on

Temporal Databases

by

Jan Chomicki
Monmouth University

David Toman
BRICS

List of Slides

Temporal databases

Databases with a **time** dimension.

Examples:

- various kinds of records:
  - ⇒ credit
  - ⇒ personnel
  - ⇒ financial
  - ⇒ judicial
  - ⇒ cadastral (property)
  - ⇒ medical
- monitoring and measurement results
- ...

As current data grows stale, it becomes **historical** data.

Plan

- Abstract Temporal Data Models and Query Languages
- Practical Temporal Models and Query Languages
- More Powerful Languages
- Incomplete Information in Temporal Databases
- Temporal Integrity Constraints
- Research Problems

The structure of time

Temporal **ontology**:

- databases: points (instants)
- AI: intervals (periods)

Temporal **domain** T:

- a linearly-ordered set
- typically one of the well known mathematical domains: **N** (natural numbers, **Z** (integers), **Q** (rationals), **R** (reals)

There is also a data domain D of uninterpreted constants.

Ontology important because it determines the kind of temporal object facts are associated with.

Domain properties (discreteness, density, ...) also important.

Skipped in this tutorial

Nonlinear temporal domains:

- branching time has potential database applications in versioning and workflows [Attie et al., 1993]
- proposals too preliminary

Multiple time granularities:

- important practical issue
- many possible approaches
  [Ladkin, 1986, Wang et al., 1993, Bettini et al., 1995]

Multiple temporal dimensions

To model multiple kinds of time:

- valid time vs. transaction time

To represent intervals using pairs of points.

To represent multiple temporal attributes in query results.

Plan

- **Abstract Temporal Data Models and Query Languages**
  - $\Rightarrow$ **Data models**
  - $\Rightarrow$ **Query languages**
  - $\Rightarrow$ **Expressive power**
- Practical Temporal Models and Query Languages
- More Powerful Languages
- Incomplete Information in Temporal Databases
- Temporal Integrity Constraints
- Research Problems

Abstract Temporal Databases

Representation-independent semantics of temporal databases.

Advantages:

- allows high-level, declarative query languages
- provides a formal framework to solve outstanding problems in temporal databases:
  - ⇒ interoperability of different data models
  - ⇒ functional dependencies and normal forms

Explicitly introduced in [Chomicki, 1994] but implicit in many recent papers (*snapshot-equivalence*).

Basic Building Blocks

- Temporal domain $(T, <)$ (linearly ordered, unbounded)
- Relational databases $DB(D, \rho)$ over $(D, =)$ and schema ρ:

$$(\quad \underbrace{D, =,}_{\text{domain of uninterpreted constants}} \quad \underbrace{r_1, \ldots, r_n}_{\text{a finite instance of } \rho \text{ over } D} \quad)$$

- Major approaches to putting these together:
  1. the snapshot model
  2. the timestamp model

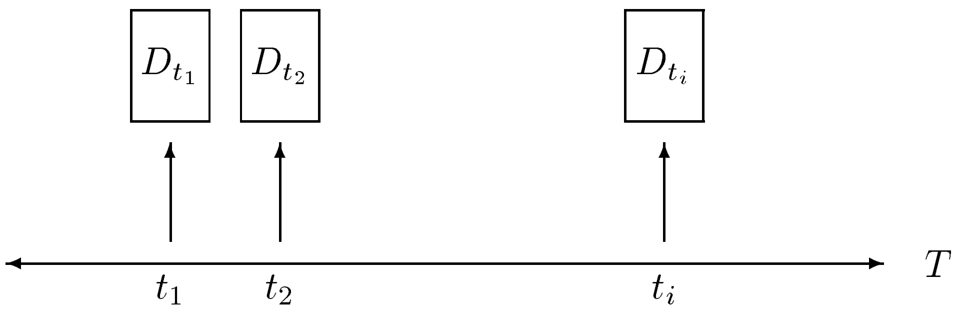
Relational databases serve as descriptions of *worlds* at every single time instant.

This is the major difference from the propositional uses of TL (e.g., in the area of specification and verification).

There are several ways of adding time (a single temporal dimension) to the relational model.

The Snapshot Model

Temporal database is a function $T \to DB(D, \rho)$



Data type of relations $T \to (D^n \to \mathrm{bool})$

This approach is the closest to Propositional TL: every database describes the state of the world at a particular time instant; the order relation $<$ on T then defines the flow of time (the access relation).

Snapshots: Example

Year	Snapshot
. . .	
1985	{Work(Mary,DEC)}
. . .	
1990	{Work(John, IBM), Work(Mary,IBM), Work(Steve,HP)}
1991	. . .
1992	. . .
1993	{ Work(John,Microsoft),Work(Mary,IBM), Work(Steve,HP)}
. . .	

Histories

Relational database updates map states to states.

History:

- a sequence of database states
- thus also a **snapshot temporal database**

The Timestamp Model

For every $r_i \in \rho$ we define a relation R_i such that

$$R_i = \{(t, a_1, \ldots, a_k) : (a_1, \ldots, a_k) \in r_i \text{ in } D_t\}$$

A *Timestamp Temporal Database* is a structure

$$(\quad \underbrace{D, =,}_{\text{domain of uninterpreted constants}} \quad \underbrace{T, <,}_{\text{temporal domain}} \quad \underbrace{R_1, \ldots, R_n}_{\text{an instance of } \rho \text{ over } T \text{ and } D} \quad)$$

Data type of relations $(T \times D^n) \to \mathrm{bool}$

The snapshot model is not very suitable for queries of the form

$$\{t : DB \models \varphi(t)\}$$

(all time instants such that φ holds in DB).

The alternative *timestamp model* makes time to be just another data type (however, we must remember that it comes with an interpreted order relation).

The two models are equivalent (type-isomorphic).

Timestamp Example

Work		
Name	Company	Year
John	IBM	1990
John	IBM	1991
John	DEC	1992
John	Microsoft	1993
John	Microsoft	. . .
Mary	DEC	1984
Mary	DEC	1985
Mary	IBM	1990
Mary	IBM	. . .
Steve	HP	1990
Steve	HP	. . .

Finiteness not an issue *at this level*.

Different temporal data models - different representations.

Query Languages

Start from *Relational Calculus* over $\rho \cup \{=\}$

$$L ::= r_i(x_1, \ldots, x_k) \mid x_i = x_j \mid L \wedge L \mid \neg L \mid \exists x.L$$

Example: *list everyone who works for IBM.*

$$\{x : \exists y.\mathrm{Works}(x, y) \wedge y = \mathrm{IBM}\}$$

Two principal temporal extensions:

- implicit references to time (temporal connectives)
- explicit references to time (variables and quantifiers)

First-order Temporal Connectives

Temporal Connectives:

O	$::=$	X_i	–	placeholders
		$O \wedge O$	}	propositional connectives
		$\neg O$		
		$t_i < t_j$	}	temporal variables
		$\exists t_i.O$		

Definition. A *(k-ary) temporal connective* is an O-formula with exactly one free variable t_0 and k free predicate variables $X_1, \ldots, X_k$.

The O-formulas define the *truth tables* for the new connectives.

At this point we only allow *first-order* connectives; therefore the language of O-formulas is just FO logic over the signature of the temporal domain $(T, <)$.

Examples of Temporal Connectives

The standard connectives **since** and **until**:

$$X_1 \textbf{ until } X_2 \triangleq \exists t_2 . t_0 < t_2 \wedge X_2 \wedge \forall t_1 (t_0 < t_1 < t_2 \to X_1)$$
$$X_1 \textbf{ since } X_2 \triangleq \exists t_2 . t_0 > t_2 \wedge X_2 \wedge \forall t_1 (t_0 > t_1 > t_2 \to X_1)$$

The derived connectives:

$$\Diamond X_1 \triangleq \text{true } \textbf{until } X_1 \qquad \Box X_1 \triangleq \neg\Diamond\neg X_1$$
$$\blacklozenge X_1 \triangleq \text{true } \textbf{since } X_1 \qquad \blacksquare X_1 \triangleq \neg\Diamond\neg X_1$$

For discrete linear order we also define the $\bigcirc$ (next) and $\bullet$ (previous) operators as

$$\bigcirc X_1 \triangleq \exists t_1 . t_1 = t_0 + 1 \wedge X_1 \qquad \bullet X_1 \triangleq \exists t_1 . t_1 + 1 = t_0 \wedge X_1$$

Note that all the connectives are first-order definable.

Propositional Temporal Logic

Propositional TL(**since**, **until**)

is expressively equivalent to monadic FOL($<$)
over complete linear orders [Kamp, 1968]

Propositional TL(**until**)

is equivalent to TL(**since**, **until**)
on (discrete) linear orders bounded in the past.

Formulas in Propositional TL(**since**, **until**)

can be separated to pure past, present and future parts:
$\Rightarrow$ a *boolean combination* of φ_{past}, φ_{present}, and φ_{future}.

First-order Temporal Logic: syntax

Let Ω be a finite set of temporal connectives.

$$
\begin{array}{lllll}
F & ::= & r_i(x_{i_1}, \ldots, x_{i_k}) & – & \text{database schema} \\
 & & F \wedge F & \rbrace & \text{propositional connectives} \\
 & & \neg F & & \\
 & & x_i = x_j & \rbrace & \text{data variables} \\
 & & \exists x_i . F & & \\
 & & \boxed{\omega(F_1, \ldots, F_k)} & – & \text{temporal connectives}
\end{array}
$$

Features: encapsulation, same schema as the original relational database,...

FOTL: semantics

$DB, \theta, t \models r_j(x_{i_1}, \ldots, x_{i_k})$	if $r_j \in \rho, (\theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in r_j^{DB(t)}$
$DB, \theta, t \models x_i = x_j$	if $\theta(x_i) = \theta(x_j)$
$DB, \theta, t \models \varphi \wedge \psi$	if $DB, \theta, t \models \varphi$ and $DB, \theta, t \models \psi$
$DB, \theta, t \models \neg\varphi$	if not $DB, \theta, t \models \varphi$
$DB, \theta, t \models \exists x_i.\varphi$	if there is $a \in D$ and $DB, \theta[x_i \mapsto a], t \models \varphi$
$DB, \theta, t \models \omega(F_1, \ldots, F_k)$	if $T_P, [t_0 \mapsto t] \models \omega^*$ where $T_P, \delta \models X_i$ iff $DB, \theta, \delta(t_i) \models F_i$

Answer: $\varphi(DB) = \{t, \theta : DB, \theta, t \models \varphi\}$.

Examples

- *John's work history*

$$\exists x.\mathrm{Works}(x, y) \wedge x = \mathrm{John}$$

- *all people who were rehired by the same company*

$$\exists y.\mathrm{Works}(x, y) \wedge \blacklozenge(\neg\mathrm{Works}(x, y) \wedge \blacklozenge\mathrm{Works}(x, y))$$

- *all people who have been unemployed between two jobs*

$$\exists y.\mathrm{Works}(x, y) \wedge \blacklozenge(\neg\exists y.\mathrm{Works}(x, y) \wedge \blacklozenge\exists y.\mathrm{Works}(x, y))$$

- *all people who continuously worked at IBM since John left*

$$\mathrm{Works}(x, \mathrm{IBM}) \ \mathbf{since} \ (\mathrm{Works}(\mathrm{John}, \mathrm{IBM}) \wedge \square\neg\mathrm{Works}(\mathrm{John}, \mathrm{IBM}))$$

Temporal Relational Calculus: syntax

$$
\begin{array}{lll}
M ::= & \boxed{R_i(t_j, x_{i_1}, \ldots, x_{i_k})} & \text{– extended database schema} \\
& M \wedge M & \multirow{2}{*}{\text{propositional connectives}} \\
& \neg M & \\
& x_i = x_j & \multirow{2}{*}{\text{data variables}} \\
& \exists x_i . M & \\
& \boxed{\begin{array}{l} t_i < t_j \\ \exists t_i . M \end{array}} & \text{temporal variables}
\end{array}
$$

⇒ essentially a two-sorted first-order logic (2-FOL)

Temporal RC: Semantics

$DB, \theta \models R_j(t_i, x_{i_1}, \ldots, x_{i_k})$	if $R_j \in \rho, (\theta(t_i), \theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in R_j^{DB}$
$DB, \theta \models t_i < t_j$	if $\theta(t_i) < \theta(t_j)$
$DB, \theta \models x_i = x_j$	if $\theta(x_i) = \theta(x_j)$
$DB, \theta \models \varphi \wedge \psi$	if $DB, \theta \models \varphi$ and $DB, \theta \models \psi$
$DB, \theta \models \neg\varphi$	if not $DB, \theta \models \varphi$
$DB, \theta \models \exists t_i.\varphi$	if there is $s \in T_P$ and $DB, \theta[t_i \mapsto s] \models \varphi$
$DB, \theta \models \exists x_i.\varphi$	if there is $a \in D$ and $DB, \theta[x_i \mapsto a] \models \varphi$

Answer: $\varphi(DB) = \{\theta : DB, \theta \models \varphi\}$.

Examples

- *John's work history*

$$\exists x.\mathrm{Works}(t_0, x, y) \wedge x = \mathrm{John}$$

- *all people who were rehired by the same company*

$$\exists y.\mathrm{Works}(t_0, x, y) \wedge \exists t_1 (t_1 < t_0 \wedge \neg \mathrm{Works}(t_1, x, y) \wedge \\ \exists t_2 . t_2 < t_1 \wedge \mathrm{Works}(t_2, x, y))$$

Examples (cont.)

- *all people who have been unemployed between two jobs*

$$\exists y.\mathrm{Works}(t_0, x, y) \wedge \exists t_1 (t_1 < t_0 \wedge \neg \exists y.\mathrm{Works}(t_1, x, y) \wedge \\ \exists t_2 . t_2 < t_1 \wedge \exists y.\mathrm{Works}(t_2, x, y))$$

- *all people who continuously worked at IBM since John left*

$$\exists t_1 < t_0\, (\mathrm{Works}(t_1, \mathrm{John}, \mathrm{IBM}) \wedge \\ \forall t_2 . t_2 > t_1 \to \neg \mathrm{Works}(t_2, \mathrm{John}, \mathrm{IBM}) \wedge \\ \forall t_2 . t_1 < t_2 \leq t_0 \to \mathrm{Works}(t_2, x, \mathrm{IBM}))$$

Expressive Power

$$
\begin{aligned}
\mathrm{Embed}(r_i(x_1,\ldots,x_{v_i})) &= R_i(t_0,x_1,\ldots,x_{v_i})\\
\mathrm{Embed}(x_i = x_j) &= x_i = x_j\\
\mathrm{Embed}(F_1 \wedge F_2) &= \mathrm{Embed}(F_1) \wedge \mathrm{Embed}(F_2)\\
\mathrm{Embed}(\neg F) &= \neg\,\mathrm{Embed}(F)\\
\mathrm{Embed}(\exists x.F) &= \exists x.\,\mathrm{Embed}(F)\\
\mathrm{Embed}(\omega(F_1,\ldots,F_k)) &= \\
&\omega^*(\mathrm{Embed}(F_1)[t_0/t_1],\ldots,\mathrm{Embed}(F_k)[t_0/t_k])
\end{aligned}
$$

Expressive Power (cont.)

1. FOTL is strictly less expressive than 2-FOL
   for arbitrary finite set of connectives.

$$\exists t_1 < t < t_2 . \forall x (P(t_1, x) \iff R(t_2, x))$$

2. Future FOTL is less expressive than FOTL
   for linear order with left endpoint

$$\exists t > 0 . \forall x (P(0, x) \iff R(t, x))$$

3. FOTL = 2-FOL
   if the data domain D has fixed size

How do we prove it?

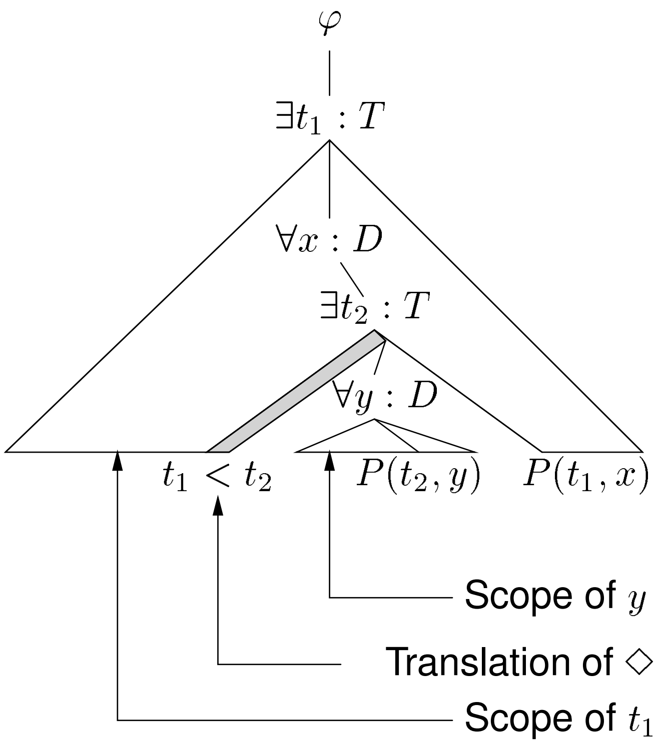
Solution #1:

- we formulate a syntactic restriction on variables names that can occur together in the leaves of FOTL formulas,
- we modify the Ehrenfeucht-Fraïssé Game to capture exactly this restriction, and
- we show structures (temporal databases) that are distinguishable by a 2-FOL formula but equivalent w.r.t. the Ehrenfeucht-Fraïssé Game.

Solution #2:

- Communication Complexity [Abiteboul et al., 1996]

Scope of Temporal Variables



Ehrenfeucht-Fraïssé Games

(1) Game on structures $\mathcal{A}$ and $\mathcal{B}$, common signature
(2) Fixed set of variables $\{v_1, \ldots, v_k\}$
(3) Two vectors $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$

$\mathcal{A}$ and $\mathcal{B}$ can be distinguished by

- quantifier-free formulas over $\{v_1, \ldots, v_k\}$ if

  $\Rightarrow$ $\mathcal{A} \models R[a_1, \ldots, a_k]$ and $\mathcal{B} \not\models R[b_1, \ldots, b_k]$ or

  $\Rightarrow$ $\mathcal{A} \not\models R[a_1, \ldots, a_k]$ and $\mathcal{B} \models R[b_1, \ldots, b_k]$

- $\forall x.\varphi$ with free variables $\{v_1, \ldots, v_k\}$ if $\exists a \in A\ \forall b \in B$

  $$\mathcal{A} \not\models \varphi[a_1, \ldots, a_k, a] \text{ and } \mathcal{B} \models \varphi[b_1, \ldots, b_k, b]$$

  or vice versa.

Ehrenfeucht-Fraïssé Games (cont.)

- Players PLAYER I and PLAYER II play n moves

  "PLAYER I picks an element from A or B and then PLAYER II picks an element from B or A"

- Result: n pairs

$$(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$$

- Winning condition for PLAYER II:

$$R^{\mathcal{A}}(a_{i_1}, \ldots, a_{i_k}) \iff R^{\mathcal{B}}(b_{i_1}, \ldots, b_{i_k})$$

  $\Rightarrow$ the map $a_i \mapsto b_i$ is a partial iso.

- Winning strategy for PLAYER II defines

  an equivalence relation $\mathcal{A}; a \sim_{k,n} \mathcal{B}; b$

Ehrenfeucht-Fraïssé Games (cont.)

Theorem

$$\mathcal{A}; a \sim_{k,n} \mathcal{B}; b \quad \text{iff} \quad \forall \varphi \in \mathcal{L}_n . \mathcal{A}; a \models \varphi \iff \mathcal{B}; b \models \varphi$$

Method

$\mathcal{A}_n, \mathcal{B}_n$ a sequence of pairs of structures such that:

(1) $\mathcal{A}_n \in P$ and $\mathcal{B}_n \notin P$

(2) $\mathcal{A}_n \sim_n \mathcal{B}_n$

for all $n > 0$. Then P is not definable.

Ehrenfeucht-Fraïssé Games (cont.)

Facts:

- In $Th(=)$ we can not distinguish finite sets such that

$$card(S) \geq qd(\varphi)$$

- In $Th(<_{Lin})$ we can not distinguish finite sets such that

$$card(S) \geq 2^{qd(\varphi)-1}$$

- Connectivity is not FO definable.

EF Games and Temporal Logic

How do we use EF-games for Temporal logic?

- trying to show sub-FO expressive power
  $\Rightarrow$ the equivalence generated by the game
  has to be stronger
  $\Rightarrow$ weaker winning condition for $\textsc{Player II}$.
- we base the winning condition on the limited scope of temporal variables on the FOTL formulas
  $\Rightarrow$ variable compatibility criterion

Compatibility of Variables in FOTL

$$
\begin{array}{l|l|l}
\text{Round of the game:} & A & t_0\ a_1\ a_2\ t_3\ t_4\ a_5\ a_6\ t_7\ a_8\ \ldots\ t_i\ \ldots\ a_n \\
 & B & s_0\ b_1\ b_2\ s_3\ s_4\ b_5\ b_6\ s_7\ b_8\ \ldots\ s_i\ \ldots\ b_n \\
\hline
\text{Compatible moves:} & 0 & \\
 & 3 & \\
 & 4 & \\
 & 7 & \\
 & \vdots & \vdots \\
 & i & \\
 & \vdots & \vdots
\end{array}
$$

New winning condition for $\textsc{Player II}$**:**

⇒ check for partial iso. only for *compatible* moves.

Databases not distinguishable by FOTL

- Databases $\mathcal{A}, \mathcal{B}$ with two relations: P and R

  $\mathcal{A}$ $P(t) \in C(k,n)$, $S \in C(k,n) \supset \exists t. S = P(t)$ $R(t) \in C(k,n) - K$,
  $S \in C(k,n) - K \supset \exists t. S = R(t)$

  $\mathcal{B}$ $P(t) \in C(k,n)$, $S \in C(k,n) \supset \exists t. S = P(t)$
  $R(t) \in C(k,n)$, $S \in C(k,n) \supset \exists t. S = R(t)$

  where $C(k,m)$ is the set of all k element subsets of M, $M = \{0, \ldots, m-1\}$, $K = \{0, \ldots, k-1\}$, and $P(t) = \{x : P(t,x)\}$ (similarly for R).

- Formula: $\varphi = \forall t \exists s \forall x. P(t,x) \supset R(s,x)$
  $\Rightarrow \mathcal{A} \not\models \varphi$ and $\mathcal{A} \models \varphi$

- $\mathcal{A} \sim_{1,k} \mathcal{B}$

Communication Complexity

- Every FO(E)TL formula has
  a *constant communication complexity* (on split databases)

  The sequence $\pi_1, \pi_2, \ldots, \pi_r$ of all *temporal* subformulas of φ such that if π_i is a subformula of π_j then $i < j$ defines a communication protocol with *constant complexity* k, r.

- Equality, Inclusion, and Disjointness

  $\Rightarrow$ do not have constant communication complexity

  $\Rightarrow$ are expressible in 2-FOL

Consequences for Temporal Queries

- There is no FO-complete relational algebra closed over uniform temporal databases.
  1. weaker query languages
  2. non-uniform temporal databases
  3. not closed query languages
- Nested queries can *not* be decomposed into a sequence of (not nested) views.

Temporal Relational Algebra

A Temporal Relational Algebra (TRA) consists of:

- Universe: single-dimensional temporal relations
- Operations: a finite set of operations of the type

$$2^{T\times D^{n_1}} \times \ldots \times 2^{T\times D^{n_k}} \to 2^{T\times D^{n}}$$

- First-order operations: can be defined by a first-order formula.

All operations preserve closure over the universe
$\Rightarrow$ can *not* express all FO queries.

The problem is not in the operators, but in the *closure* requirement—the universe contains only single-dimensional temporal relations.

Major problem from the implementation point of view: a limitation to single-dimensional temporal relations prevents expressing all FO queries.

However, the majority of practical temporal query languages ignore this fact!

TRA: example

- Universe: temporal relations of type $T \times D^k$.
- Operations:

$$
\begin{aligned}
\pi_V(R) &= \{t, \theta_{|V} : DB, \theta, t \models R\} \\
\sigma_F(R) &= \{t, \theta_{|FV(R)} : DB, \theta, t \models R \wedge F\} \\
R \bowtie S &= \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \wedge S\} \\
R \cup S &= \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \vee S\} \\
R - S &= \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \wedge \neg S\} \\
\mathcal{S}(R, S) &= \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \;\mathbf{since}\; S\} \\
\mathcal{U}(R, S) &= \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \;\mathbf{until}\; S\}
\end{aligned}
$$

Temporal Logic TL(FO)

FOTL does not have the separation property of the propositional logic:

$$\forall x. \blacklozenge p(x) \iff \Diamond p(x)$$

Weaker two-layer logic TL(FO) [Gabbay et al., 1994]:

$\Rightarrow$ temporal connectives only outside of the scopes of $\forall$ and $\exists$.
$\Rightarrow$ has separation property

Plan

- Abstract Temporal Data Models and Query Languages
- **Practical Temporal Models and Query Languages**
  - ⇒ **Compact encoding of temporal databases**
  - ⇒ **Query languages over the encodings**
  - ⇒ **Query languages and translations**
  - ⇒ **Efficient relational operations**
- More Powerful Languages
- Incomplete Information in Temporal Databases
- Temporal Integrity Constraints
- Research Problems

Concrete Temporal Databases

Grouping by non-temporal attributes

Finite encodings of the resulting sets of time instants

$$
\underset{T \times D^k}{\boxed{\begin{array}{cc} \multicolumn{2}{c}{\ldots} \\ t_1, & a_1, \ldots, a_k \\ t_2, & a_1, \ldots, a_k \\ \vdots & \vdots \\ t_k, & a_1, \ldots, a_k \\ \multicolumn{2}{c}{\ldots} \end{array}}}
\quad \longrightarrow \quad
\underset{2^T \times D^k}{\boxed{\begin{array}{cc} \multicolumn{2}{c}{\ldots} \\ \{t_1, t_2, \ldots, t_k\}, & a_1, \ldots, a_k \\ \multicolumn{2}{c}{\ldots} \end{array}}}
$$

Implicit functional dependency $A_1 \ldots A_k \to T$ in $2^T \times D^k$.

Observation:
a single (data) tuple is usually related to a large number of time instants

Finite Encoding using Constraints

- sets of time instants described by their characteristic formulas
- the language of constraints admits quantifier elimination
- fixed-size tuples = quantifier-free formulas in CNF

$$\boxed{\begin{array}{c} \dots \\ \{t_1, t_2, \dots, t_k\}, \quad a_1, \dots, a_k \\ \dots \end{array}} \longrightarrow \boxed{\begin{array}{c} \dots \\ \varphi(t), \quad a_1, \dots, a_k \\ \dots \end{array}} \longrightarrow \boxed{\begin{array}{cc} \multicolumn{2}{c}{\dots} \\ \varphi_1, & a_1, \dots, a_k \\ \vdots & \vdots \\ \varphi_l, & a_1, \dots, a_k \\ \multicolumn{2}{c}{\dots} \end{array}}$$

Interval Encoding

Let T_P be a linearly ordered temporal domain. We define a set

$$I(T) = \{(a,b) : a \leq b, a \in T \cup \{-\infty\}, b \in T \cup \{\infty\}\}$$

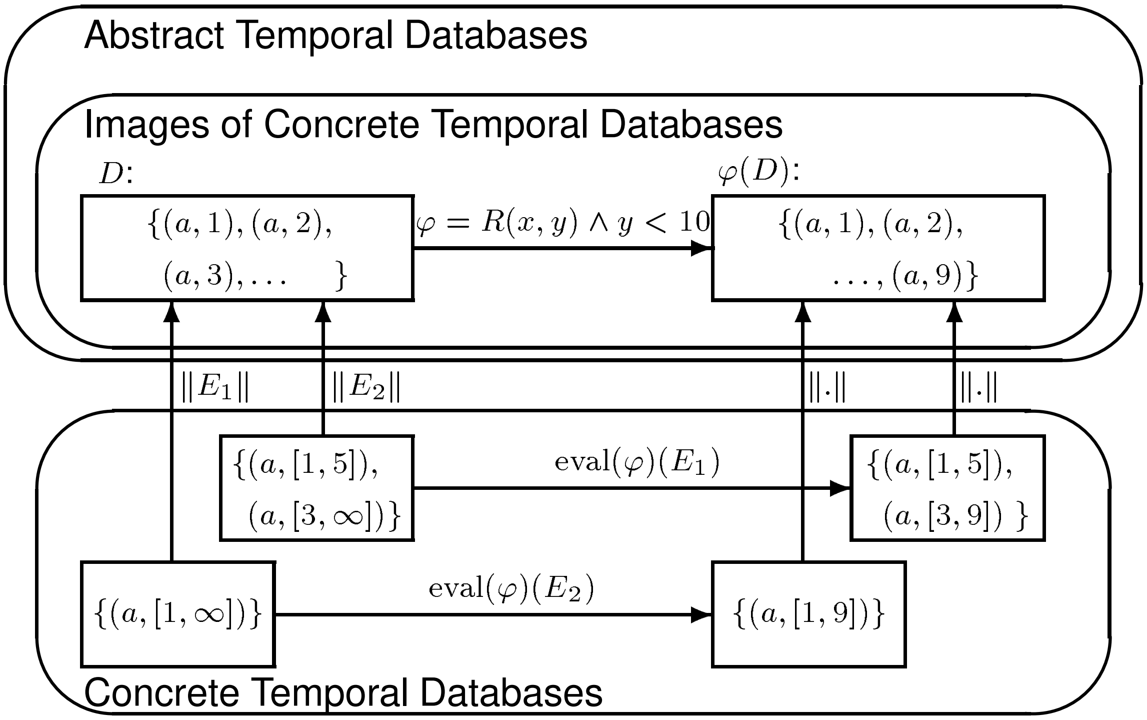
Relations on the elements of $I(T)$:

$$([a,b] <_{--} [a',b']) \iff a < a' \quad ([a,b] <_{+-} [a',b']) \iff b < a'$$
$$([a,b] <_{-+} [a',b']) \iff a < b' \quad ([a,b] <_{++} [a',b']) \iff b < b'$$

The structure $T_I = (I(T), <_{--}, <_{+-}, <_{-+}, <_{++})$ is the *Interval-based Temporal Domain (corresponding to T_P)*.

Most prominent example of constraint encoding: quantifier free formulas in the theory of (linear) order with constants.

Interval Encoding (cont.)



Example

Work		
Name	Company	Year
John	IBM	[1990, 1991]
John	DEC	[1992, 1992]
John	MicroSoft	[1993, ∞]
Mary	DEC	[1984, 1985]
Mary	IBM	[1990, ∞]
Steve	HP	[1990, ∞]

Why Intervals?

- Natural description of time *periods*

  $\Rightarrow$ common in *real world*

- Also a natural consequence of **updates** of standard relational databases (see *Histories*)

  $\Rightarrow$ *insertion* of tuple a at time t corresponds to insertion of tuple(s) $([t,\infty],a)$ to the corresponding history.

  $\Rightarrow$ *deletion* of tuple a at time t corresponds to deletion of tuple(s) $([t,\infty],a)$ from the corresponding history.

Interval Queries

$$
\begin{array}{lllll}
M & ::= & R_i(I_j, x_{i_1}, \ldots, x_{i_k}) & - & \text{extended database schema} \\
 & & M \wedge M & \multirow{2}{*}{\}} & \multirow{2}{*}{\text{propositional connectives}} \\
 & & \neg M & & \\
 & & x_i = x_j & \multirow{2}{*}{\}} & \multirow{2}{*}{\text{data variables}} \\
 & & \exists x_i . M & & \\
 & & I_i^* < I_j^* & \multirow{2}{*}{\}} & \multirow{2}{*}{\text{temporal variables}} \\
 & & \exists I_i . M & &
\end{array}
$$

L^I is the language of M-formulas.

Note that I's range over $I(T)$ – INTERVALS.

Genericity

Definition. $\varphi \in L_I$ is $\|.\|$-*generic* if

$$\|D_1\| = \|D_2\| \supset \|\varphi D_1\| = \|\varphi D_2\|$$

for all D_1, D_2 concrete temporal databases.

Let D_1, D_2 be two concrete temporal databases such that:

$$\begin{aligned} R^{D_1} &= \{([0,3], a)\} \\ R^{D_2} &= \{([0,2], a), ([1,3], a)\} \end{aligned}$$

Then

$$\exists I, J.\exists x(R(I,x) \wedge R(J,x) \wedge I \neq J)$$

is true in D_2 but false in D_1 $\rightarrow$ not generic.

TSQL2 [Snodgrass, 1995]

- similar extension to SQL
- valid and transaction time:
  $\Rightarrow$ temporal elements as timestamps
  $\Rightarrow$ tuples can not be bounded by their arity
- no formal semantics
  $\Rightarrow$ unclear expressive power

Duplicate Semantics

- standard SQL semantics

  ⇒ select-project-join, aggregation, difference, union
  ⇒ infinite bags w/finite duplication

- no duplicate-preserving projection of unbounded attributes:

  ⇒ `select all` name `from indep`

  Result:

$$\{\ldots,\underbrace{(\text{Poland}),\ldots,(\text{Poland})}_{\infty},\ldots\}$$

- duplicate preserving projection of *bounded* types may be allowed

The semantics of the new language is exactly SQL semantics on the $\|.\|$-images (the semantics is naturally extended to infinite relations).

We need to restrict projections to maintain finite duplication (infinite duplication is not very interesting nor useful).

Note the difference between infinite and unbounded (we can "measure" infinite bounded sets, e.g., in dense case; we may get non-integral "counts", e.g., 1.5 years)

Doesn't work after coalescing; aggregation often based on the encoding!

TSQL2's Successors

1. ATSQL
2. SQL/Temporal

Common characteristics:

- *interval ontology* of time rather than point-based ontology
- still 1 or 2 dimensional
- allow coercion of temporal attributes to data attributes and vice versa
- introduces levels of compatibility with SQL
  $\Rightarrow$ separate semantics for each level

Example

list all people who were rehired by the same company

```
select  r.name
from    Works r, Works s
where   r.name=s.name
  and   r.company=s.company
  and   validtime(r) precedes validtime(s)
```

Does not work in general:

⇒ `indep` has to be coalesced and can not contain duplicates.

Coalescing

A single-dimensional temporal relation is *coalesced* if every fact is associated only with maximal non-overlapping intervals.

- coalescing has to be enforced using a non-logical operation on relations
- relational operations do not preserve coalescing in general e.g., projection, union, or set difference.

A way to curb non-genericity in the single-dimensional case.

Example (cont.)

list all people who were unemployed (between jobs):

```
select  r.name
from    Works r, Works s
where   r.name=s.name
  and   validtime(r) precedes validtime(s)
```

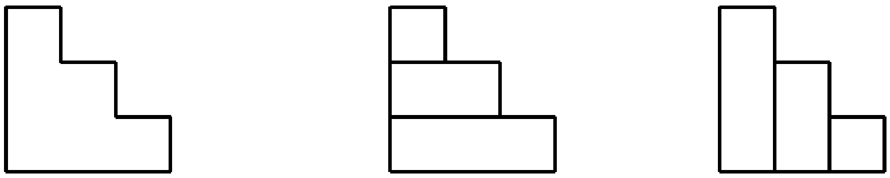
Again, this doesn't work:

`indep` has to be recoalesced **after projecting out** company
⇒ to get correct answer we have to use subqueries/views.

Failure of Coalescing

Known facts:

- there is no unique coalescing for temporal dimension > 1
- FO-complete query languages can not avoid arbitrarily large dimensionality in queries
  [Abiteboul et al., 1996, Toman and Niwinski, 1996]



- queries depend on the representation
  $\Rightarrow$ non-$\|.\|$-generic queries

Folding and Unfolding

Definition Let R be a concrete temporal relation. Then

$$\mathrm{unfold}(R) = \{(t, !x) : \exists I. R(I, !x) \wedge t \in I\}.$$

IXRM [Lorentzos, 1993]

- interval, multidimensional approach
- unfold
  $\Rightarrow$ EXPSPACE for finite intervals
  $\Rightarrow$ not defined for unbounded intervals
- can't be used as a basis for query evaluation

Other Proposals

HRDM [Clifford and Croker, 1987]

time-dependent attributes (functions of time instants)

TQUEL [Snodgrass, 1987] introduces two temporal dimensions:

$\Rightarrow$ valid time (when a fact is *true*)
$\Rightarrow$ transaction time (when a fact is *recorded in the DB*)

interval encoding and coalescing
item only PSJ queries directly represented in the syntax

TSQL [Navathe and Ahmed, 1993]

interval encoding of 1-dimensional time line
based on temporal relational algebra

Intervals vs. True Intervals

- Intervals as encodings of sets of time instants:

```
king("Charles IV", "Czech Kingdom",[1347,1378])
king("Casimir III","Poland",        [1333,1370])
```

- Intervals as points in two-dimensional space:

```
electricity("Jones", 40, 05/15/96, 06/15/96)
electricity("Smith", 35, 05/01/96, 06/01/96)
```


Moral: don't mix apples and oranges!

Temporal Connectives in $\mathrm{L_I}$

- Coalescing simplifies select-project-join queries
- However, it is of little help even for elementary *temporal* operations:

$$A \textbf{ since } B \mapsto [\max(A^-, succ(B^-)), A^+]$$
for $\max(A^-, succ(B^-)) \leq A^+$ and $B^+ \geq A^-$

$$A \textbf{ until } B \mapsto [A^-, \min(A^+, pred(B^+))]$$
for $A^- \leq \min(A^+, pred(B^+))$ and $A^+ \geq B^-$

For details see [Böhlen et al., 1996].

It is still fairly difficult to state even elementary temporal queries.

Translations

Another approach:

- use an *abstract* query language
- devise an evaluation procedure over encoded databases

In our case we consider the following two pairs:

1. FOTL + interval encoding [Böhlen et al., 1996]
2. 2-FOL + interval encoding [Toman, 1996, Toman, 1997]

The target language is L^I (in the first case with coalescing).

Closure for Intervals

Queries must preserve **closure**.

Example:

```
select   r.Name, r.Year, s.Year
from     Work r, Work s
where    r.Name = s.name
  and    r.Year < s.Year
```

Produces a *triangle-like* sets of time instants in two-dimensional plane.

- can't be allowed (as the result can't be stored)
- has to be *evaded* during query evaluation

$\Rightarrow$ *Attribute Independence*

Can all queries be asked over interval encoding? NO: the results must be representable as concrete relations. Similar to safety restrictions.

The attribute independence is required *only* for the attributes in the answer, not for all attributes in the query.

Note that most temporal query languages allow only a single temporal attribute in the answers: in these cases the condition is trivial (true).

SQL/TP [Toman, 1997]

Temporal attributes range over individual time instants

- Syntax and Semantics:
  ⇒ an extension of SQL by a new data type
  ⇒ supports finite duplication
- Encoding of relations:
  ⇒ compact representation of temporal relations (intervals)
  ⇒ syntactic restrictions to maintain closure (safety)
- Query evaluation:
  ⇒ efficiency: depends on the (size of the) encoding only
  ⇒ allows compilation to SQL/92

The goals:

(1) The syntax and semantics of standard SQL has to be extended in a *natural way*: by adding a new data type `time` that behaves (almost) identically as the existing data types, and

(2) The syntax and semantics are *independent* of the chosen encoding of timestamps.

The chosen language is derived from calculus with finite duplicates.

The choice of encoding is not critical: we aim on subsuming TSQL, so we have chosen TSQL's encoding of temporal databases.

Efficient of query evaluation means that it uses only elements in the active domain of the encoding (plus some small neighborhoods of `Time` values).

SQL/TP: syntax

```
<exp>     ::= <ddl_exp> | <dql_exp> | <dml_exp>

<ddl_exp> ::= create table <rid> ( <signat> ) |
              create view  <rid> ( <dql_exp> )

<dql_exp> ::= select [all] <s_list> [from <f_list>]
                     [where <cond>] [group by <g_list>] |
              ( <dql_exp> ) union [all] ( <dql exp> ) |
              ( <dql_exp> ) except [all] ( <dql exp> ) |
              ( <dql_exp> ) intersect [all] ( <dql exp> )

<dml_exp> ::= insert [all] into <rid> ( <dql_exp> ) |
              delete [all] from <rid> ( <dql_exp> )
```


Standard SQL, all nesting of queries is realized in the `from` clause.

The `having` clause and nesting in the `where` clause are easily definable using nesting in the `from` clause and thus are considered to be just a syntactic sugar.

Where is the “temporal extension”? In the data definition language: when we define an attribute of type `time` (next slide). . .

SQL/TP: encoding of time

`<type> ::= time [using <enc>] | integer | char(N) | ...`

Modifiers `<enc>`:

- `points`
- `bounded intervals`
- `unbounded intervals`

Other classes of constraints can be readily incorporated

`points` is for atomic events (e.g., transaction time).

`bounded intervals` is for facts with bounded "duration".

`unbounded intervals` is for facts with unbounded duration (e.g., x holds from now on)

Easy to introduce other constraint classes (back-door for the CDB folks): If you can't beat them (SQL), join them.

SQL/TP: Query Evaluation

$$\text{compile} : \text{SQL/TP} \longrightarrow \text{SQL/92}$$

$$\begin{tikzcd}
\boxed{\text{Abstract TDB}} \arrow[d, "Q_{\text{SQL/TP}}"] & \boxed{\text{Concrete TDB}} \arrow[l, "\|\cdot\|"'] \arrow[d, "Q'_{\text{SQL/92}} = \text{compile}(Q)"] & \boxed{\text{Physical DB}} \arrow[l] \arrow[d, "\text{eval}(Q')"] \\
\boxed{\text{Abstract Relation}} & \boxed{\text{Concrete Relation}} \arrow[l, "\|\cdot\|"'] & \boxed{\text{Physical Relation}} \arrow[l]
\end{tikzcd}$$

So far we only hinted about the query evaluation of SQL/TP queries over the concrete databases (note that the semantics is defined over the abstract databases.

We chose compilation: easy implementation on top of an existing RDBMS.

Also we identify operations that we may want to add to a RDBMS to facilitate more efficient evaluation of temporal queries.

Data Definition Language

```
create table indep ( Name char(20), year time )
```

$\big\downarrow$ compile

```
create table indep ( Name char(20),
                     yearmin Time, yearmax Time)
```


The data definition language is extremely easy to translate: we just replace the abstract signature with its concrete counterpart.

`Time` is an UDT (based on integers or a `DATE` type of the DBMS).

How do we compile Queries?

Goal: to translate a query where *temporal attributes range over individual time instants* to *an equivalent* query where *temporal attributes range over interval endpoints*.

Steps:

- translation through **conditional queries** to evade closure problems
- use of **quantifier elimination** on point-based temporal attributes
- uses of a **normalization** to encode set operations

Closure for SQL/TP

Syntactic (safety) restrictions for SQL/TP queries:

- top level signature: all attributes have to be pairwise independent [Chomicki et al., 1996]
- aggregated attributes independent of grouping attributes
- no duplicate preserving projections of unbounded attributes

Conditional Queries

Subqueries of a SQL/TP query may not be *attribute independent*:

⇒ We use *conditional queries* $Q[\varphi]$

- Q is a SQL/92 query
- φ is a quantifier free formula in $\mathcal{L}_<$

Translation Invariant:

$$\text{compile}(Q) = \{Q_1[\varphi_1], \ldots, Q_n[\varphi_n]\}$$
$$\text{such that } Q(\|D\|) = \sigma_{\varphi_1}\|Q_1(D)\| \cup \ldots \cup \sigma_{\varphi_n}\|Q_n(D)\|$$
$$\text{and } \mathcal{L}_< \not\models \varphi_i \wedge \varphi_j,\ 0 < i < j \leq n$$

While the top level temporal attributes have to be independent, this may not be true for all the subqueries. The conditional formulas allow us to propagate and eventually eliminate such dependencies (they are constraints "global" for the whole subquery: in a CDB approach such a constraint would be attached to every tuple in the answer to the subquery).

"compile" is defined by structural induction on the syntax of SQL/TP queries in such a way that it preserves the above invariant.

Select Block: Join and Selection

Czech Kingdom	1098 1620

$\times$

Poland	1025 1794
Slovakia	1940 1944

$\Downarrow$ $\Downarrow$

Czech Kingdom	1098 1620	Poland	1025 1794
Czech Kingdom	1098 1620	Slovakia	1940 1944

$\Downarrow \sigma_{n_1='\text{Czech Kingdom}', t_1\mathtt{min} \le t_2\mathtt{max}, t_2\mathtt{min} \le t_1\mathtt{max},}$

Czech Kingdom	1098 1620	Poland	1025 1794

Select block is translated in two steps: first we translate the product–selection part (cf. query in Example 1).

The selection condition on the temporal attributes is derived using quantifier elimination of the point attributes from

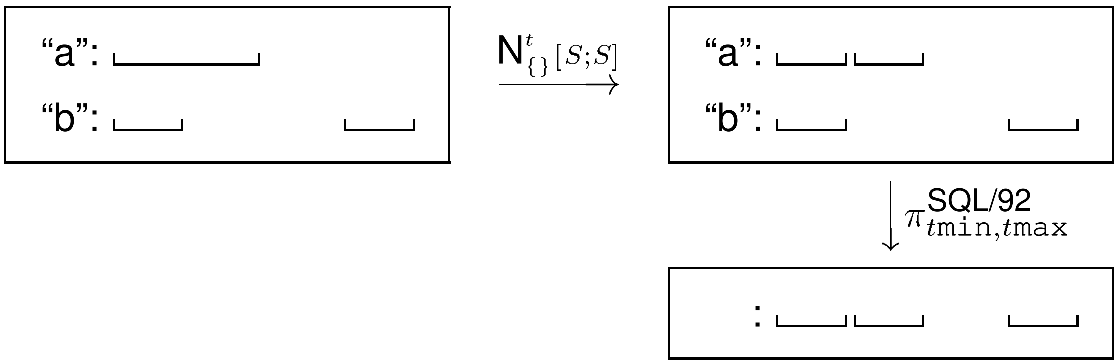
$$\exists t_1, t_2 . t_1\mathtt{min} \le t_1 \le t_1\mathtt{max} \wedge t_2\mathtt{min} \le t_2 \le t_2\mathtt{max} \wedge t_1 = t_2$$

The input tables may be results to subqueries: we need to add the conditional formulas associated with these subqueries.

The product-selection is converted to joins by the underlying DBMS as for any other query.

Also we derive a resulting condition by eliminating all but the point temporal attributes (in this case $t_1 = t_2$).

Select Block: Duplicate Elimination



The second step is the projection, duplicate elimination (and aggregation, if needed).

The top left and right tables are $\|.\|$-equivalent.

The standard projection with duplicate elimination applied on the top right table produces the desired result.

The normalization N guarantees that the set/bag operation of SQL (e.g. projection with duplicate elimination) works correctly.

Time-compatible Queries

Let $\{Q_1, \ldots, Q_k\}$ be a set of queries (compatible signatures), X a subset of their data attributes, and t a temporal attribute.

$Q_1, \ldots, Q_k$ are t*-compatible on* X if

$$a \in Q_i(D) \text{ and } b \in Q_j(D) \text{ such that } \pi_X(a) = \pi_X(b)$$
$$\text{then } \pi_{\{t\}}(\|a\|) \text{ and } \pi_{\{t\}}(\|b\|) \text{ are identical or disjoint.}$$

$Q_1, \ldots, Q_k$ are *time-compatible* on X if $Q_1, \ldots, Q_k$ are t-compatible on X for all temporal attributes t in the common signature.

Normalization

Lemma: There are first order queries $\mathsf{N}_X^t[Q_i; Q_1, \ldots, Q_k]$ such that

(1) $\|Q_i(D)\| = \|\,\mathsf{N}_X^t[Q_i; Q_1, \ldots, Q_k](D)\|$
(2) $\{\mathsf{N}_X^t[Q_i; Q_1, \ldots, Q_k] : 0 < i \leq k\}$ are t-compatible

The t-compatibility guarantees that the interval values of t behave like points: they are identical when the representation is identical and disjoint when the representation differs.

This approach can (and is) used for all set/bag operations. It also allows the alternative use of merge-joins (cf. intersection) in the `from` clause (this is too tricky to be found by the underlying DBMS).

Set Operations

Q_1 `<setop>` Q_2 is translated using the following Lemma:

Lemma. Let $\{Q_1, Q_2\}$ be time-compatible. Then

$$\|Q_1\| \ \mathtt{op} \ \|Q_2\| = \|Q_1 \ \mathtt{op} \ Q_2\|$$

where $\mathtt{op} \in \{\cap, \cup, -\}$ is a set operator on relations.

$\Rightarrow$ compile$(Q_1 \ \mathtt{op} \ Q_2) =$ N$[Q_1; Q_1, Q_2] \ \mathtt{op} \ N[Q_2; Q_1, Q_2]$
$\Rightarrow$ in $O(n \log n)$ of the combined size of Q_1 and Q_2

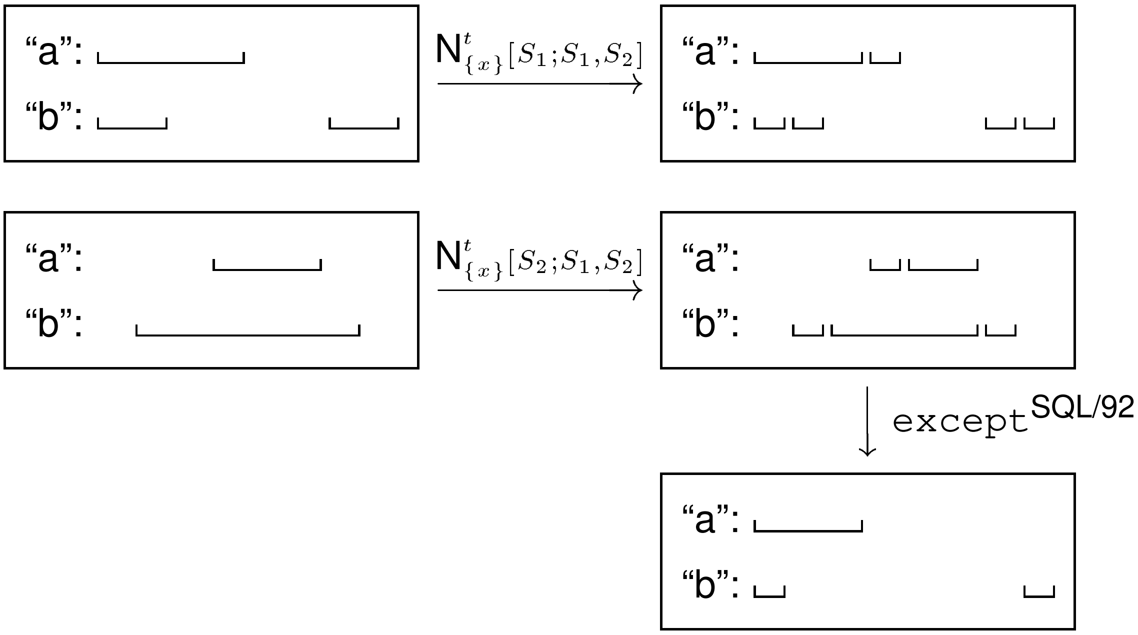
The set operations are based on N similarly to duplicate elimination.

N not necessary in the case of additive union (`union all`).

The intersection technique can be used for merge-joins in the translations of the select block.

N can be based on sorting: the cost may amortize in subsequent join and duplicate elimination steps.

Set Operations (cont.)



This example shows how a difference is computed using N.

Note that it preserves duplicates, if necessary.

Set Operations (cont.)

$$\{Q_i[\varphi_i]\}\ \texttt{union}\ \{R_j[\psi_j]\} \mapsto$$
$$\{Q_i[\varphi_i \wedge \neg \textstyle\bigwedge_j \psi_j], R_j[\psi_j \wedge \neg \textstyle\bigwedge_i \varphi_i], Q_i \cup R_j[\varphi_i \wedge \psi_j]\}$$

$$\{Q_i[\varphi_i]\}\ \texttt{except}\ \{R_j[\psi_j]\} \mapsto$$
$$\{Q_i[\varphi_i \wedge \neg \textstyle\bigwedge_j \psi_j], Q_i - R_j[\varphi_i \wedge \psi_j]\}$$

$$\{Q_i[\varphi_i]\}\ \texttt{intersect}\ \{R_j[\psi_j]\} \mapsto$$
$$\{Q_i \cap R_j[\varphi_i \wedge \psi_j]\}$$

The conditional queries are handled using the above rules: the goal is to push the set operation towards the unconditional parts of the queries.

All the conditional formulas are computed at compile time.

This transformation is at most quadratic (not exponential).

Size of the Translated Query

- the size of the result:

$$|\,\text{compile}(Q)| \in O(2^{|Q|})$$

$\Rightarrow$ exponential in depth (nesting) of the query.

- views:

```
create view <rid> as ( <query> )
```

$\Rightarrow$ views have to obey attribute independence
$\Rightarrow$ for queries that can be decomposed into *small* views:

$$|\,\text{compile}(Q)| \in O(|Q|)$$

This is slightly unpleasant. However, the exponential nature is in the depth of nesting, which is usually quite low.

Moreover, there aren't too many conditional formulas in low dimensions. Essentially, the conditional formulas define hyper-stripes parallel to the diagonal, with an offset defined by a constant present in the query.

Also for large classes of queries (e.g., queries equivalent to a query in TSQL, TRA, or FOTL) the exponential blowup can be avoided altogether.

Optimization

1. reduce the number of conditional queries:

   ⇒ combine queries with the same condition can in a single query:

   $$Q_1[\varphi], Q_2[\varphi] \to Q_1 \cup Q_2[\varphi]$$

   - for 1-dimensional time: the only condition is true
   - for 2-dimensional time: $t_1 < t_2 + c$
     ⇒ diagonal stripes

2. eliminate redundant N operations:

   $Q_1, \ldots, Q_k$ are t-compatible on X
   $\Rightarrow Q_i(D) = \mathsf{N}[Q_i; Q_1, \ldots, Q_k](D)$

3. eliminate redundant duplicate elimination.

(1) The merging is possible due to the observation on the previous slide: there are relatively few different conditional formulas.

(2) N is (almost) idempotent (similar to rules for projection). thus we can use algebraic rewrites to limit the use of N to the necessary minimum (also we could have maintain the base relations normalized (w.r.t. frequent queries) and this way shift the burden of normalization to updates. We have rules that tell us how the t-compatibility interacts with relational operators.

(3) In many cases we do not have to perform duplicate elimination (however, its more complex than in the SQL case). We can detect these cases and remove unnecessary duplicate elimination steps automatically (this is also the reason for having the `select` clause to eliminate duplicates by default).

Physical Query Processing

Efficient execution of queries on the chosen encoding.

⇒ General constraint encoding [Kanellakis et al., 1993].

⇒ Interval encoding:

- equality on time instants maps into interval intersection

  ⇒ indices for intervals

- set operations on points map to

  ⇒ normalization followed by the original operation

  ⇒ on-the-fly multidimensional indices

Intervals (or hypercubes) are important in general: more complex encodings base indexing on *bounding boxes*.

Indexing in Temporal Databases

Problem:

given an interval I, retrieve all tuples whose timestamp intersects I.

Additional knowledge:

- intervals are nonempty (thus not all pairs of time instants form an interval)
- "current" state may be accessed by far more queries than the other states of the database
- temporal databases are often append-only

Leads to unbalanced access structures.

Indexing (cont.)

- Time Index [Elmasri et al., 1990, Ramaswamy, 1997]
- R-Tree [Guttman, 1984]

  general multi-dimensional index
  used in POSTGRESS [Stonebraker and Kemnitz, 1991]
- AP-Tree

  append-only index
  combination of ISAM and B^+-tree
- ...

Survey of indexing methods: [Saltzberg and Tsotras, 1994]

Indexing (example)

Time Index [Elmasri et al., 1990]:

- at every interval start- or end-point store the list of all intervals containing this point
- build a B-tree over the above points

Queries in $O(\log n + s/B)$.

Disadvantage: $O(n^2/B)$ space requirement
$\Rightarrow$ [Ramaswamy, 1997] improves the method to space $O(n/B)$
(amortized updates!)

Updates in Temporal Databases

Depend on choice of semantics: sets vs. bags.

Insertions

$\Rightarrow$ duplicate tuples and overlapping timestamps

Deletions

$\Rightarrow$ may have to delete only part of a concrete tuple
$\Rightarrow$ set-based deletions

Updates

$\Rightarrow$ similar to deletes: parts of concrete tuples
$\Rightarrow$ problem: updates *in-place*

Plan

- Abstract Temporal Data Models and Query Languages
- Practical Temporal Models and Query Languages
- **More Powerful Languages**
  - ⇒ **multidimensional connectives**
  - ⇒ **higher-order features**
- Incomplete Information in Temporal Databases
- Temporal Integrity Constraints
- Research Problems

More Powerful Languages

Additional expressive power:

- More temporal dimensions
- Higher-order temporal connectives
- Higher-order features in the underlying language

Multidimensional Temporal Logics

- A *k-ary m-dimensional temporal connective* is a formula in the first-order language of the temporal domain T_P with exactly m free variables $t_0^1, \ldots, t_0^m$ and k free predicate variables $X_1, \ldots, X_k$
- The language $L^{\Omega(m)}$ is a first-order logic extended with a finite set $\Omega(m)$ of m-dimensional temporal connectives.
- Semantics defined using satisfaction relation with m evaluation points:

$$DB, \theta, t_1, \ldots, t_m \models \varphi$$

Multidimensional TL (example)

- Two-dimensional logic

$$\begin{aligned} \Diamond &= \exists t_1.X_1 \\ \uparrow\downarrow &= \exists t_1.(t_1[1] = t_0[2] \wedge t_1[2] = t_0[1] \wedge X_1) \end{aligned}$$

- other examples:
  - ⇒ the temporal logic with the *now* [Kamp, 1971] operator,
  - ⇒ the Vlach and Åqvist system [Åqvist, 1979], and
  - ⇒ most of the interval logics [Allen, 1984].
- k-FOTL$\sqsubset$ $(k+1)$-FOTL [Toman and Niwinski, 1996]

Why?

- Formula

$$\forall t_0, \ldots, t_k \exists s \forall x.(P_0(t_1, x) \vee \ldots, \vee P_0(t_k, x)) \supset R(s, x)$$

  is in $(k+1)$-FOTL but not in k-FOTL.

- proof:

  ⇒ Ehrenfeucht-Fraïssé Games modified for k-FOTL

    dense linear order
    integer order

  ⇒ Communication complexity

    not clear how to use

Higher-order features

1. in the temporal dimension:
   - ETL (connectives use regular exps) [Wolper, 1983]
   - fixed point extensions (μTL) [Vardi, 1988]
   - second order logic (in $<$) for defining temporal connectives
2. in the data dimension (stratified w.r.t. temporal connectives):
   - Datalog($^{\neg}$)
   - fixed-point extensions of relational calculus
3. in both:
   - Datalog$_{1S}$ and TempLog [Baudinet et al., 1993]

Datalog$_{1S}$ and Templog

"Find all the computers at risk where "being at risk" is defined in the following way: a computer is at risk *at a given time if it has been earlier infected or it has been in contact with a computer already at risk.*

$$atRisk(X, T+1) \leftarrow infected(X, T).$$
$$atRisk(X, T+1) \leftarrow atRisk(X, T).$$
$$atRisk(X, T+1) \leftarrow contacted(X, Y, T), atRisk(Y, T).$$

This is Datalog$_{1S}$, the Templog formulation is similar.

A query that is inductive on both *time* and *data*.

New application area of Datalog$_{1S}$ and extensions: operational semantics of active databases [Motakis and Zaniolo, 1997].

Plan

- Abstract Temporal Data Models and Query Languages
- Practical Temporal Models and Query Languages
- More Powerful Languages
- **Incomplete Information in Temporal Databases**
  - $\Rightarrow$ **Motivation**
  - $\Rightarrow$ **Marked Nulls and Constraints**
  - $\Rightarrow$ **Queries**
- Temporal Integrity Constraints
- Research Problems

Incomplete temporal information

Partial information:

- *Sue stopped working for Microsoft and started to work for IBM before 1992*
- *John worked for IBM before working for Microsoft*

Different granularities:

- *The Beatles broke up in the sixties*

More natural here than in many other database applications.

Null values

SQL-92 NULL is not enough:

- stands for *“value unknown”*
- lack of consistent formal interpretation in queries
- unable to express order, succession and other logical conditions

Marked nulls

The model of Imieliński and Lipski:

- marked nulls
- local conditions associated with tuples
- global conditions associated with tables

Semantics of a table:

- set of relations (possible worlds)

Constraint databases with marked nulls

[Koubarakis, 1994]:

- two kinds of variables: universally and existentially quantified
- conditions: *(gap)-order constraints*.

Two-stage formal semantics:

- substitute time values for nulls in such a way that the global condition is satisfied, obtaining a set of generalized relations $rep(R)$
- for every element of $rep(R)$ take the corresponding abstract temporal database

This works for an arbitrary number of temporal dimensions.

One dimension: one can use *indefinite intervals*, together with global conditions.

Example

Work		
Name	Company	Year
John	IBM	$1990 \leq Year \leq c_1$
John	Microsoft	$c_2 \leq Year$
Sue	Microsoft	$Year \leq c_3$
Sue	IBM	$c_3 < Year$
Global condition		
$1990 < c_1 < c_2 \ \wedge \ c_3 < 1992$		

One possible world

Work		
Name	Company	Year
John	IBM	1990
John	Microsoft	1991, 1992, ...
Sue	Microsoft	..., 1991
Sue	IBM	1992, 1993, ...

Another possible world

Work		
Name	Company	Year
John	IBM	1990
John	Microsoft	1993, 1994, ...
Sue	Microsoft	..., 1989
Sue	IBM	1990, 1991, ...

There are more possible worlds.

Query languages

Relational languages:

- (extended) relational algebra
- (extended) relational calculus

Modal operators:

- possibility: *“Is it possible that John and Sue overlapped at IBM?”*
- certainty: *“Is it certain that John worked at IBM in 1990?”*

Both calculus and algebra are extended with the operators.

Query evaluation

Relational calculus:

- translation to relational algebra
- quantifier elimination (for modal queries)

Complexity results for the general model [Koubarakis, 1997]:

- not worse than for incomplete *relational* databases

Tractability results for a restricted model [van der Meyden, 1997]

- local conditions = equalities
- any practical temporal domain
- crucially depends on the *number of temporal dimensions*:
  - $\Rightarrow$ PTIME for one dimension,
  - $\Rightarrow$ co-NP-complete for more than one dimension.

Other approaches

Temporal databases:

- point and interval relationships [Chaudhuri, 1988]
- quantitative indeterminacy [Dyreson and Snodgrass, 1993]
- lower and upper bounds on an interval [Gadia et al., 1992]

Artificial intelligence:

- 13 basic kinds of relationships between intervals, algebra of relationships [Allen, 1983]
- disjunctive information
- metric information [Meiri, 1991], [Kautz and Ladkin, 1991]

Example: $I_1\{\mathbf{before}, \mathbf{after}\}I_2$ equivalent to $I_1^+ < I_2^- \vee I_2^+ < I_1^-$

The *qualitative* part of these can be recast using Koubarakis' approach.

Plan

- Abstract Temporal Data Models and Query Languages
- Practical Temporal Models and Query Languages
- More Powerful Languages
- Incomplete Information in Temporal Databases
- **Temporal Integrity Constraints**
  - ⇒ **Functional Dependencies and Normal Forms**
  - ⇒ **Constraint Dependencies**
  - ⇒ **Temporal Integrity Constraints in Relational DBs**
- Research Problems

Temporal Integrity Constraints

Relational databases: integrity constraints (ICs) are closed first-order logic formulas.

Temporal databases: integrity constraints are closed formulas in a first-order temporal query language.

Why first order: evaluated over *first order structures*.

Applications of ICs

Integrity constraints capture the semantics of a database application.

Enforcing database integrity:

- only meaningful information is stored

Database design:

- normal forms (criteria for *good*, anomaly-free schemas)
- *good* decompositions

Functional Dependencies (FDs)

The most popular class of dependencies, essential for defining **keys** and **normal forms**.

Example: relation schema `Emp(SSN,Name,Salary)`.

The FD $\texttt{SSN} \rightarrow \texttt{Salary}$ holds in `Emp` if for every instance r of `Emp`, corresponding to a possible state of the real world, and for every pair of tuples $t_1, t_2 \in r$:

if $t_1[\texttt{SSN}] = t_2[\texttt{SSN}]$, then $t_1[\texttt{Salary}] = t_2[\texttt{Salary}]$.

$$\forall s_1, s_2, n_1, n_2, d_1, d_2 . \texttt{Emp}(s_1, n_1, d_1) \wedge \texttt{Emp}(s_2, n_2, d_2) \wedge s_1 = s_2 \Rightarrow d_1 = d_2$$

Can be generalized to *sets* of attributes in both sides.

FDs in Database Design

A set of attributes X is a **key** of a relation schema U w.r.t. a given set of FDs F if:

1. for every attribute $A \in U$, the FD $X \to A$ is implied by F, and
2. no proper subset of X has property 1.

A relation schema U is in **Boyce-Codd Normal Form** (BCNF) w.r.t. a set of FDs F if for every dependency $X \to A$ implied by F, X is a superset of a key of U.

FDs in temporal databases

Snapshot functional dependencies:
⇒ should hold in every **snapshot**.

Example: relation schema `Emp(SSN,Name,Salary,Year)`

Snapshot dependency is a classical FD:

$$\texttt{SSN Year} \rightarrow \texttt{Salary}$$

Compare with:

$$\texttt{SSN} \rightarrow \texttt{Salary}$$

which means that the salary never changes.

The classical relational database design theory is still applicable.

Enough to consider temporal extensions of relation schemas with one (or more) temporal attributes.

Keys, normal forms, decompositions etc. - as in relational databases.

Temporal functional dependencies

Many attempts to define temporal FDs for concrete temporal databases:

- latest and most general [Jensen et al., 1996]
- no logical formulation for dependencies
- technical problems [Chomicki, 1997]

The approach of Jensen, Snodgrass and Soo is subsumed, generalized, and clarified by the classical approach [Chomicki, 1997].

Constraint Dependencies

Functional dependencies are **equality-generating.** In temporal databases it is natural to consider more general **constraint-generating** dependencies [Baudinet et al., 1995].

Example: *"The past never changes."*

$$\forall x, x', t_v, t'_v, t_t, t'_t.\ p(x, t_v, t_t) \wedge p(x', t'_v, t'_t) \wedge t_v < t'_v \Rightarrow t_t < t'_t$$

Valid time: t_v, transaction time t_t.

Many examples in [Jensen and Snodgrass, 1994].

Complexity results for the implication problem [Baudinet et al., 1995].

Not clear how to use CGDs for database design.

Further generalizations: tuple-generating constraint dependencies [Maher and Srivastava, 199

Temporal ICs in Relational DBs

Integrity constraints on **histories.**

Examples:

"salaries of employees should never decrease"

"employees cannot be hired if they have been fired in the past"

"a recalled book should be returned within a week"

Constraint Satisfaction

A history H **satisfies** a constraint C
if C is true in every state of H.

A finite history H **potentially satisfies** C if it can be extended to an infinite history that satisfies C.

Implementation: after each update check whether C is potentially satisfied in the current history.

More flexible enforcement: temporal triggers.

Constraints in Temporal Logic

Many variants proposed and studied.

"employees can not be hired if they have been fired in the past"

$$\neg(\exists x)(hired(x) \wedge \blacklozenge fired(x))$$

"employees can not be hired if they have been fired in the past and not subsequently reinstated"

$$\neg(\exists x)(hired(x) \wedge (\neg reinstated(x) \;\mathbf{since}\; fired(x)))$$

"employees can be reinstated at most once"

$$\neg(\exists x)(reinstated(x) \wedge \blacklozenge reinstated(x))$$

Biquantified Formulas

[Lipeck and Saake, 1987]

- only **future** connectives
- quantifiers either **external** (not in the scope of any temporal connective) or **internal** (no temporal connective in their scope)
- no internal quantifiers: potential constraint satisfaction decidable in EXPTIME [Chomicki and Niwinski, 1995]
- one internal quantifier: undecidable.

Not surprising, in view of the fact that first-order temporal logic is highly undecidable.

Past Formulas

[Chomicki, 1995]

- only past connectives
- arbitrary quantifiers
- potential constraint satisfaction undecidable
- a practical method that approximates potential constraint satisfaction:

  $\Rightarrow$ check if the constraint is satisfied in the current state

Also *real-time* temporal logic.

History Encoding

Encoding in [Chomicki, 1995]:

- the database schema is augmented by *auxiliary* relations
- every database state is extended with the instances of auxiliary relations to form an *extended* state
- the extended state H'_k encodes the whole current history $(H_0, \ldots, H_k)$.

Properties of the encoding:

1. Incrementally computable.
2. Lossy.
3. Space-efficient.

Auxiliary Relations

Auxiliary relations for a constraint C:

- one auxiliary relation r_α for each temporal subformula α of C (i.e., of the form $\bullet A$ or $A \mathbf{\ since\ } B$),
- the arity of r_α is equal to the number of free variables in α,
- auxiliary constraint relation r_C (0-ary).

Auxiliary relations are defined inductively (induction on time). The definitions are automatically derived from the constraints. The relations are implemented as materialized views.

The constraint C is satisfied iff the constraint relation r_C contains the empty tuple $()$.

the auxiliary relation definitions are based on recurrent definitions of the past temporal connectives:

$$\varphi \mathbf{\ since\ } \psi = (\varphi \wedge \bullet\psi) \vee (\varphi \wedge \bullet(\varphi \mathbf{\ since\ } \psi))$$

Example

"employees can not be hired if they have been fired in the past and not subsequently reinstated"

$$\neg(\exists x)(hired(x) \wedge (\neg reinstated(x) \ \mathbf{since} \ fired(x)))$$

Auxiliary relations:

$$\begin{aligned} r_\alpha(x) &\stackrel{df}{=} \neg reinstated(x) \ \mathbf{since} \ fired(x) \\ r_C &\stackrel{df}{=} \neg(\exists x)(hired(x) \wedge r_\alpha(x)) \end{aligned}$$

Inductive definition of r_α:

$$\begin{aligned} r_\alpha^0(x) &\stackrel{df}{=} False \\ r_\alpha^{k+1}(x) &\stackrel{df}{=} (r_\alpha^k(x) \vee fired^k(x)) \wedge \neg reinstated^{k+1}(x) \end{aligned}$$

The superscript is the consecutive number of the state.

The definitions do not depend on the specific value of the superscript, provided it is greater than 0.

The definitions are implemented as materialized views, using an active DBMS [Chomicki and Tor

Space and time requirements

Assumptions:

- a fixed finite set of integrity constraints IC,
- a fixed database schema.

The active domain $adom(D)$ of a history $D = (D_0, \ldots, D_k)$ is the set of domain values that appear in D.

Definition: A history encoding is *bounded* if:

1. the extended schema is fixed and finite.
2. for every $n \in N$ there is an $m \in N$ such that for every $H = (H_0, \ldots, H_k)$, the following condition is satisfied:

   if $|adom(H)| < n$, then $size(H'_k) < m$.

Bounded vs. Unbounded Encoding

Example:

	0	1	2	3	$\cdots$
$\{x : p(x)\}$	$\{a\}$	$\{\}$	$\{a\}$	$\{\}$	$\cdots$
$\{x : \blacklozenge p(x)\}$	$\{\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\cdots$

The auxiliary relation requires constant space.

	0	1	2	3	$\cdots$
$\{x : p(x)\}$	$\{a_0\}$	$\{a_1\}$	$\{a_2\}$	$\{a_3\}$	$\cdots$
$\{x : \blacklozenge p(x)\}$	$\{\}$	$\{a_0\}$	$\{a_0, a_1\}$	$\{a_0, a_1, a_2\}$	$\cdots$

The auxiliary relation requires unbounded space because the active domain is unbounded.

Space Efficiency

Published encodings:

- [Chomicki, 1995]: bounded (polynomially)
  $\Rightarrow$ implementation [Chomicki and Toman, 1995]
- [Lipeck and Saake, 1987]: bounded
- [Sistla and Wolfson, 1995]: unbounded

Time requirements.

Of course, not only databases over fixed active domains! However, the space savings of the encoding are the most spectacular when the same domain values appear in many different database states.

Plan

- Models of time
- Abstract Temporal Data Models and Query Languages
- Practical Temporal Models and Query Languages
- More Powerful Languages
- Incomplete Information in Temporal Databases
- Temporal Integrity Constraints
- **Research Problems**

Theory

EF-Games for FOTL and richer theories of time

can the EF-Games be made less sensitive to signature extensions (like the communication complexity approach)?

Communication Complexity and k-dimensional TL

can the Communication Complexity techniques be used to show separation in dimension > 1
(note that 2-FOTL doesn't nave *constant comm. Complexity*)?

Temporal Aggregation and non-FO features in TL

can FOTL-like query languages be fitted with non-FO features (like aggregation) *nicely*?

Theory (cont.)

Concrete TDBs: Richer Encodings

- attribute independence: what's the price for syntactic closure safety?
- richer constraint theories for encoding (closure? complexity? . . .)

Bounded Encodings of the Past

- can we use more than past-FOTL?
- whats the space/time complexity?

Implementation

Prototype of SQL/TP

implementation on top of RDBMS (DB2)

Optimization Techniques for SQL/TP

- use of standard indices (whats the gain/loss)
- use of special indices, indexing for constraints

Efficient Representation of Constraint encodings

data structures, access methods, . . .

Summary Queries for Histories in Active DBMS

implementation using triggers/ECA rules

References

[Åqvist, 1979] Åqvist, L. (1979). A conjectured axiomatization of two-dimensional reichenbachian tense logic. *J. Philosophical Logic*, 8:1–45.

[Abiteboul et al., 1996] Abiteboul, S., Herr, L., and den Bussche, J. V. (1996). Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM Symposium on Principles of Database Systems*, Montréal, Canada.

[Allen, 1983] Allen, J. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843.

[Allen, 1984] Allen, J. (1984). Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154.

[Attie et al., 1993] Attie, P. C., Singh, M., Sheth, A., and Rusinkiewicz, M. (1993). Specifying and Enforcing Intertask Dependencies. In *International Conference on Very Large Data Bases*, pages 134–145.

[Baudinet et al., 1993] Baudinet, M., Chomicki, J., and Wolper, P. (1993). Temporal Deductive Databases. In [Tansel et al., 1993], pages 294–320.

[Baudinet et al., 1995] Baudinet, M., Chomicki, J., and Wolper, P. (1995). Constraint-Generating Dependencies. In *International Conference on Database Theory*, Prague, Czech Republic. Springer-Verlag. Short version in: Proc. 2nd Workshop on Principles and Practice of Constraint Programming, 1994.

[Bettini et al., 1995] Bettini, C., Wang, X., Bertino, E., and Jajodia, S. (1995). Semantic Assumptions and Query Evaluation in Temporal Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 257–268, San Jose, California.

[Böhlen et al., 1996] Böhlen, M., Chomicki, J., Snodgrass, R., and Toman, D. (1996). Querying TSQL2 Databases with Temporal Logic. In *International Conference on Extending Database Technology*, Avignon, France. Springer Verlag, LNCS 1057.

[Chaudhuri, 1988] Chaudhuri, S. (1988). Temporal Relationships in Databases. In *International Conference on Very Large Data Bases*.

[Chomicki, 1994] Chomicki, J. (1994). Temporal Query Languages: A Survey. In Gabbay, D. and Ohlbach, H., editors, *Temporal Logic, First International Conference*, pages 506–534. Springer-Verlag, LNAI 827.

[Chomicki, 1995] Chomicki, J. (1995). Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186.

[Chomicki, 1997] Chomicki, J. (1997). "Temporal" Considered Harmful in Temporal Database Design. In preparation.

[Chomicki et al., 1996] Chomicki, J., Goldin, D., and Kuper, G. (1996). Variable Independence and Aggregation Closure. In *ACM Symposium on Principles of Database Systems*, pages 40–48, Montréal, Canada.

[Chomicki and Niwinski, 1995] Chomicki, J. and Niwinski, D. (1995). On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535.

[Chomicki and Toman, 1995] Chomicki, J. and Toman, D. (1995). Implementing Temporal Integrity Constraints Using an Active DBMS. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):566–582.

[Clifford and Croker, 1987] Clifford, J. and Croker, A. (1987). The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *IEEE International Conference on Data Engineering*.

[Dyreson and Snodgrass, 1993] Dyreson, C. E. and Snodgrass, R. (1993). Historical Indeterminacy. In *IEEE International Conference on Data Engineering*.

[Elmasri et al., 1990] Elmasri, R., Wuu, G. T., and Kim, Y. G. (1990). The time index : An access structure for temporal data. In *International Conference On Very Large Data Bases*, pages 1–12, Palo Alto, Ca., USA. Morgan Kaufmann Publishers, Inc.

[Gabbay et al., 1994] Gabbay, D., Hodkinson, I., and Reynolds, M. (1994). *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press.

[Gadia et al., 1992] Gadia, S., Nair, S., and Poon, Y. (1992). Incomplete Information in Relational Temporal Databases. In *International Conference on Very Large Data Bases*.

[Guttman, 1984] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57.

[Jensen and Snodgrass, 1994] Jensen, C. and Snodgrass, R. (1994). Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):954–974.

[Jensen et al., 1996] Jensen, C., Snodgrass, R., and Soo, M. (1996). Extending Existing Dependency Theory to Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(4).

[Kamp, 1971] Kamp, H. (1971). Formal properties of 'now'. *Theoria*, 37:227–273.

[Kamp, 1968] Kamp, J. (1968). *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles.

[Kanellakis et al., 1993] Kanellakis, P., Ramaswamy, S., Vengroff, D., and Vitter, J. (1993). Indexing for Data Models with Constraints and Classes. In *ACM Symposium on Principles of Database Systems*.

[Kautz and Ladkin, 1991] Kautz, H. and Ladkin, P. (1991). Integrating Metric and Qualitative Temporal Reasoning. In *National Conference on Artificial Intelligence*.

[Koubarakis, 1994] Koubarakis, M. (1994). Database Models for Infinite and Indefinite Temporal Information. *Information Systems*, 19(2):141–174.

[Koubarakis, 1997] Koubarakis, M. (1997). The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases. *Theoretical Computer Science*, 171(1):89–112.

[Ladkin, 1986] Ladkin, P. (1986). Primitives and Units for Time Specification. In *National Conference on Artificial Intelligence*, pages 354–359.

[Lipeck and Saake, 1987] Lipeck, U. and Saake, G. (1987). Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems*, 12(3):255–269.

[Lorentzos, 1993] Lorentzos, N. A. (1993). The Interval-extended Relational Model and Its Application to Valid-time Databases. In [Tansel et al., 1993], pages 67–91.

[Maher and Srivastava, 1996] Maher, M. J. and Srivastava, D. (1996). Chasing Constrained Tuple-Generating Dependencies. In *ACM Symposium on Principles of Database Systems*, pages 128–138.

[Meiri, 1991] Meiri, I. (1991). Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *National Conference on Artificial Intelligence*.

[Motakis and Zaniolo, 1997] Motakis, I. and Zaniolo, C. (1997). Temporal Aggregation in Active Database Rules. In *ACM SIGMOD International Conference on Management of Data*.

[Navathe and Ahmed, 1993] Navathe, S. and Ahmed, R. (1993). Temporal Extensions to the Relational Model and SQL. In [Tansel et al., 1993], pages 92–109.

[Ramaswamy, 1997] Ramaswamy, S. (1997). Efficient indexing for constraint and temporal databases. In *International Conference on Database Theory*, pages 419–431. Springer-Verlag.

[Saltzberg and Tsotras, 1994] Saltzberg, B. and Tsotras, V. J. (1994). A Comparison of Access Methods for Time Evolving Data. Technical Report NU-CCS-94-21, College of Comp. Sci., Northeastern University.

[Sistla and Wolfson, 1995] Sistla, A. and Wolfson, O. (1995). Temporal Triggers in Active Databases. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):471–486.

[Snodgrass, 1987] Snodgrass, R. (1987). The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298.

[Snodgrass, 1995] Snodgrass, R., editor (1995). *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers.

[Stonebraker and Kemnitz, 1991] Stonebraker, M. and Kemnitz, G. (1991). The POSTGRES Next-Generation Database Management System. *Communications of the ACM*, 34(10):78–92.

[Tansel et al., 1993] Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R., editors (1993). *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings.

[Toman, 1996] Toman, D. (1996). Point vs. Interval-based Query Languages for Temporal Databases. In *ACM Symposium on Principles of Database Systems*, Montréal, Canada.

[Toman, 1997] Toman, D. (1997). Point-based temporal extensions of SQL. In *International Conference on Deductive and Object-Oriented Databases*.

[Toman and Niwinski, 1996] Toman, D. and Niwinski, D. (1996). First-order queries over temporal databases inexpressible in temporal logic. In *International Conference on Extending Database Technology*, Avignon, France.

[van der Meyden, 1997] van der Meyden, R. (1997). The Complexity of Querying Indefinite Data about Linearly Ordered Domains. *Journal of Computer and System Sciences*, 54(1):113–135.

[Vardi, 1988] Vardi, M. Y. (1988). A Temporal Fixpoint Calculus. In *ACM Symposium on Principles of Programming Languages*.

[Wang et al., 1993] Wang, X., Jajodia, S., and Subrahmanian, V. (1993). Temporal Modules: An Approach Toward Federated Temporal Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 227–236.

[Wolper, 1983] Wolper, P. (1983). Temporal Logic Can Be More Expressive. *Information and Control*, 56:72–99.