

# **Constraint Databases:**

**a lecture in honor of  
Paris C. Kanellakis**

**Dina Q. Goldin  
Brown University**

Here went the group photo from CP'95  
taken from Paris' memorial page

<http://www.cs.brown.edu/people/pck>

# Paris C. Kanellakis:

professor, Brown University, 1981-1995

here went the Paris's photo  
from his memorial page

## Database theory:

Providing formal foundations  
for database models and query languages.

## Outline

- **Database query theory**
  - an introduction
- **Constraint query languages** (KKR'90)
  - combining CLP and DB query concepts
- **Constraint query algebras** (GK'96)
  - the syntax and semantics of CQAs
  - towards polynomial time complexity
- **Making CDBs practical**
  - fast data access methods (KKVV'94)
- **Applications of CDBs**
  - time-series similarity querying (GK'95)
- **Conclusion**
  - current and future directions

## Relational Databases an introduction

- **Relation- and Tuple-based Worldview**

e	d
e <sub>1</sub>	d <sub>1</sub>
e <sub>2</sub>	d <sub>2</sub>
e <sub>3</sub>	d <sub>1</sub>
...	...

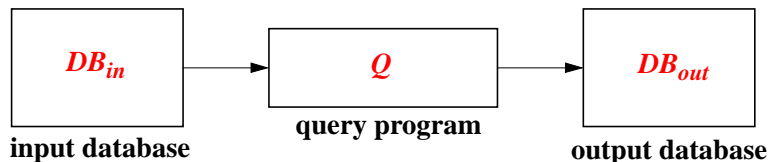
*works\_in(e, d)*

d	m
d <sub>1</sub>	m <sub>1</sub>
d <sub>2</sub>	m <sub>2</sub>
...	...

*dept\_mgr(d, m)*

*works\_in(e, d)* means “*e* works in department *d*”  
*dept\_mgr(d, m)* means “*m* manages department *d*”

- **Queries map databases to new databases**



Find relation *managed\_by(e, m)*, meaning “*m* is the manager of *e*”

## Database Query Paradigms

- **Declarative: Relational Calculus**

**queries:** function-free formulas in first-order logic  
 $\{e\ m \mid (\exists d)\ works\_in(e, d) \wedge dept\_mgr(d, m)\}$

**data:** all assignments satisfying the relational predicates  
 $works\_in(dina, cs) = TRUE$

- **Procedural: Relational Algebra**

**queries:** operator-based expressions  
 $\Pi_{e, m} (works\_in \bowtie dept\_mgr)$

**data:** sets of tuples  
 $works\_in = \{(dina, cs), (ron, custodial), \dots\}$

- **Deductive: Datalog**

**queries:** Prolog without function symbols  
 $managed\_by(e, m) :- works\_in(e, d), dept\_mgr(d, m)$

**data:** specified by Horn clauses without body  
 $works\_in(dina, cs) :- .$

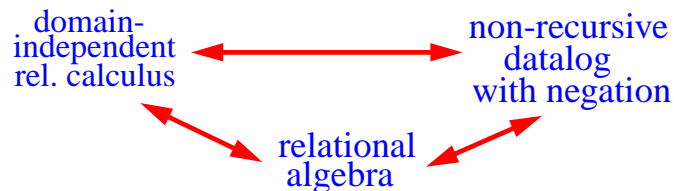
# Database Query Theory

## The data:

- A database is a finite structure containing all relevant data.
  - *finite model theory*
  - *closed-world assumption*
- The data is persistent and copious ( $|data| \gg |query|$ ).
  - *indexing structures to minimize disk accesses*
  - *data complexity measure for query analysis*

## The queries:

- A query should return all values of interest.
  - *bottom-up query semantics*
  - *query answer = complete description of the solution space*
- The output of a query over a database is also a database.
  - *the requirement of query closure*
- The three querying paradigms are equivalent.
  - *query expressibility*



# Expressibility of Transitive Closure

*Is TC expressible in first-order relational languages?*

**data:** birth records for citizens of Cambridge

**query:** how many  $n$ -th generation Cambridgians are there?

**NO! This is a PTIME-complete query**

## Increasing query expressibility

*relational calculus + recursion =  
relational algebra + fixpoint operator =  
Datalog (LP - function symbols)*

**Now, Transitive Closure is expressible.**

## Datalog as a tool in DB theory

**Challenge:** database technology should be efficient

**PCK: efficiency of Datalog programs**

time complexity  
data complexity  
parallelizability

**Challenge:** database technology should be robust

**PCK: fault-tolerance**

parallel evaluation of Datalog queries on unreliable processors

**PCK: safety of query evaluation**

type checking and unification for logic programming

**Challenge:** object-oriented databases should be theoretically sound

**PCK: theory of object-oriented databases**

Datalog + complex objects with identity and inheritance

## Database Theory vs. Programming Languages

### Logic Programming

general term  
unification  
too costly

cannot model  
many real-  
world problems



### Datalog

- no function symbols
- efficient bottom-up evaluation semantics
- cleaner DB theory

early 80's

### CLP

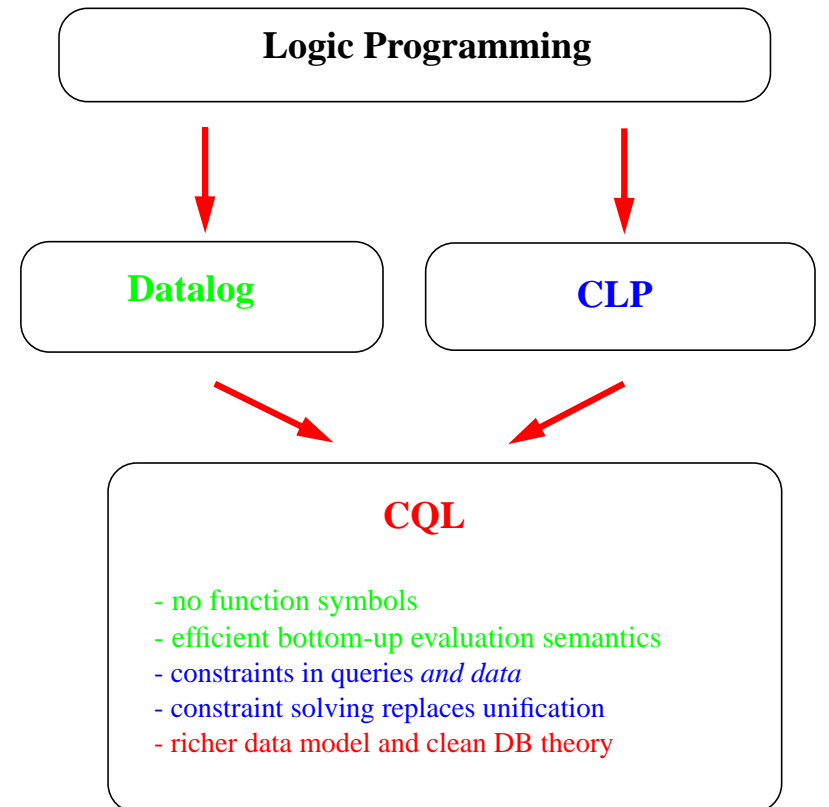
- constraints in programs
- constraint solving replaces unification
- richer data model

mid-late 80's

## The world is changing...

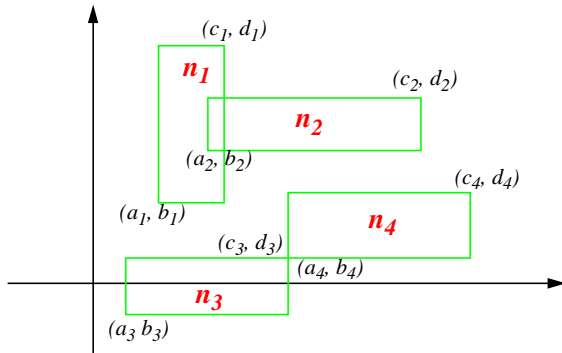
Here went the image of data coming down cables:  
administrative, images, signals...

## Merging Datalog and CLP



# Rectangle Intersection

## a spatiotemporal database example



### Relational Model:

- Each rectangle is a tuple

$Rect(n, a, b, c, d) :- n = n_i, a = a_i, b = b_i, c = c_i, d = d_i.$

- Queries contain *ad-hoc methods*

$Intsec(v_1 v_2) :- Rect(v_1, \alpha_1, \beta_1, \gamma_1, \delta_1), Rect(v_2, \alpha_2, \beta_2, \gamma_2, \delta_2),$   
 $Intersect(\alpha_1, \beta_1, \gamma_1, \delta_1, \alpha_2, \beta_2, \gamma_2, \delta_2))\}$

*query is sensitive to shape of rectangles*

# Rectangle Intersection (continued)

### Constraint model:

- Each point in each rectangle is a tuple (infinitely many). We use a **finite representation**, in the form of **constraints**.  
 $Rect\_pt(name, x, y) :- name = n_i, a_i \leq x \leq c_i, b_i \leq y \leq d_i.$
- The **semantics** of the relation is the (possibly infinite) set of all the tuples  $(Name, x, y)$  satisfying the constraints.

### Duality of representation and meaning

$Name = n_1 \wedge a_1 \leq x \leq c_1 \wedge b_1 \leq y \leq d_1$   
 $Name = n_2 \wedge a_2 \leq x \leq c_2 \wedge b_2 \leq y \leq d_2$   
 $Name = n_3 \wedge a_3 \leq x \leq c_3 \wedge b_3 \leq y \leq d_3$   
 $Name = n_4 \wedge a_4 \leq x \leq c_4 \wedge b_4 \leq y \leq d_4$

**Constraint representation of R**

Name	x	y
$n_1$	$x_1$	$y_1$
$n_1$	$x_2$	$y_2$
$n_1$	$x_3$	$y_3$
...	...	...
$n_2$	$x'_1$	$y'_1$
$n_2$	$x'_2$	$y'_2$

**The meaning of R**

- Report intersections by the query:

$Intsec(v_1 v_2) :- Rect\_pt(v_1, x, y), Rect\_pt(v_2, x, y)$

*query is insensitive to shape of data*

## Semantics of CDBs

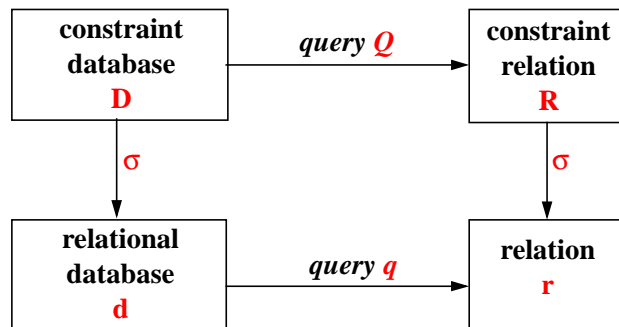
### CDB relation $R$ :

A (regular) tuple is in the semantics of  $R$  if it satisfies the constraints of some (constraint) tuple  $t$  in  $R$ .

$$\sigma(R) = \bigcup_{t \in R} P(t)$$

### CDB query $Q$ :

$q$  is a query over (regular) relations, with the semantics of *finite model theory + constraints*.



$$\sigma(Q(D)) = q(\sigma(D))$$

## CDB Theory - the basics (KKR'90)

- **Data model**

- data models are *finitely representable* (vs. finite)

- **Query closure**

- output of a query over a database is also a database *over the same constraint class*

- **Query languages**

- Declarative:** first-order formulas + *constraints*

- Deductive:** Datalog + *constraints*

- These paradigms are equivalent

- *indexing structures to minimize disk accesses*

- *data complexity measure for query analysis*

- *query answer = complete description of the solution space*

## Data Complexity of CQLs

	Relational Calculus	(Linear) Datalog with negation
Equality	$AC^0$	(NC) PTIME
Dense-order, Temporal	$AC^0$	(NC) PTIME
Real Linear	LOGSPACE (?)	-
Real Polynomial	NC (?)	-
Integer Order	PTIME (?)	-
Integer Linear	PTIME (??)	-

## Various Constraint Classes

### Equality

$$x = 3, y = 4$$

### Dense-order

$$x > 2, y \leq 3, y < x$$

### Temporal

$$y < x + 7$$

### Monotone

$$y < ax + 7, a > 0$$

### Linear

$$5y + 3x < 7, y - 2x > 9$$

### Polynomial

$$x^2 + y^2 = 4, x^2 < 3$$

## Expressibility of Parity

*Is Parity expressible in first-order + constraints?*

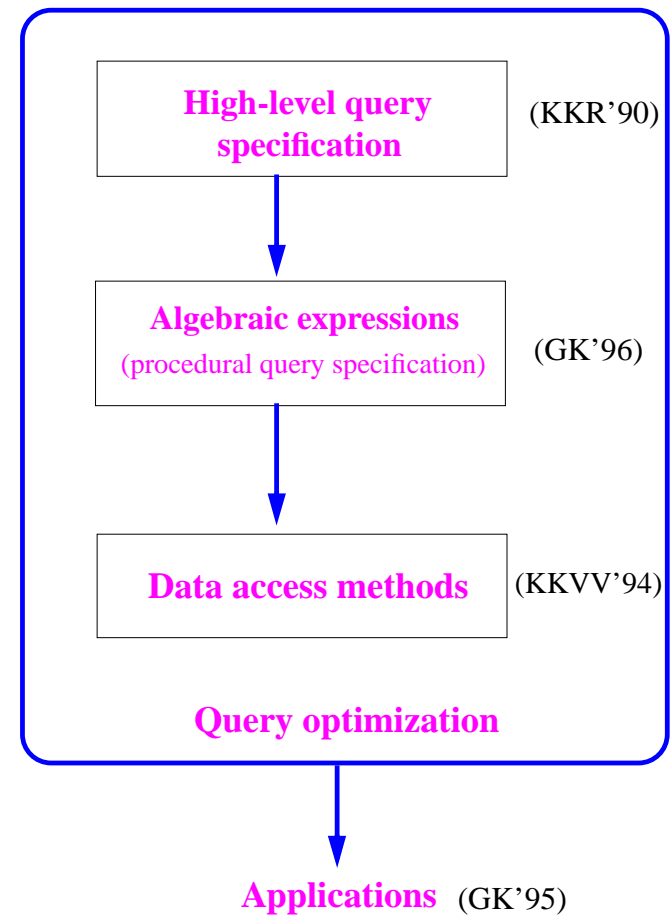
**data:** names and locations of all towns in Massachusetts

**query:** is the number of towns **even**?

***NO! Not even if the constraints are polynomial!***

PODS'96

## Database Querying: an Overview



## Database Query Algebra (GK'96)

Intermediate layer  
between high-level declarative front-end  
and low-level indexing

- **Relational algebra expressions**
  - procedural operator-based specification of the query
  - $\Pi_{e, m}(\text{works\_in} \bowtie \text{dept\_mgr})$
  - efficient translation from declarative queries and vice versa
  - bottom-up evaluation of the expression tree
- **Efficient query evaluation**
  - low data complexity
  - optimization potential

The proper context  
for studying CDB implementation issues

- **CQA implementational issues**
  - Representing the data
  - Implementing the relational operators

## Representing CDB Data

Data = Constraints

- **Store relations in a canonical form**

*minimal networks? polyhedra?*

- Ensures uniform tuple representation
- Reduces redundancy
- Improves performance of algebraic operations

- **Canonical form for dense-order constraints (GK'94)**

- Can be extended to temporal constraints
- Fast inserts and deletes
- Avoids redundancy: no empty tuples
- Projection is simple
- Useful for creating indexing structures
- Fixed size of tuple representation
- **AC<sup>0</sup>** data complexity

## CQA Operators

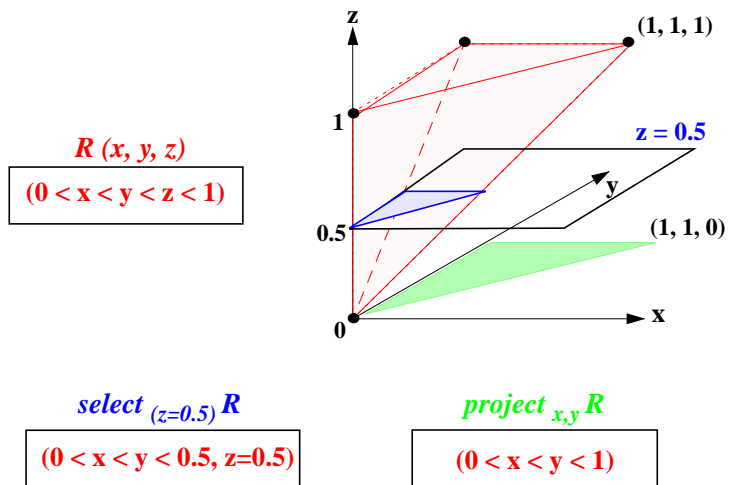
*select, project, join, union, difference, rename*

*select* all tuples in *works\_in* whose dept is CS

*project* the result onto *emp\_name* attribute

$$r1(e,d) = (works\_in(e,d) \wedge d = CS) \quad r2(e) = r1(e,d)$$

- Interpreting CQA operators



*Preserving the relational algebra syntax  
and bottom-up semantics.*

## Efficiency of CQA Operators: Projection

- Equivalent to the satisfiability problem when projecting onto 1 or 2 variables

*Weakly polynomial*

- Equivalent to **variable elimination** in the general case

*Exponential*

- Trivial for equality constraints

*Linear*

## Strongly Polynomial Projection for Monotone Constraints

$$x \leq by + c \quad (0 \leq b)$$

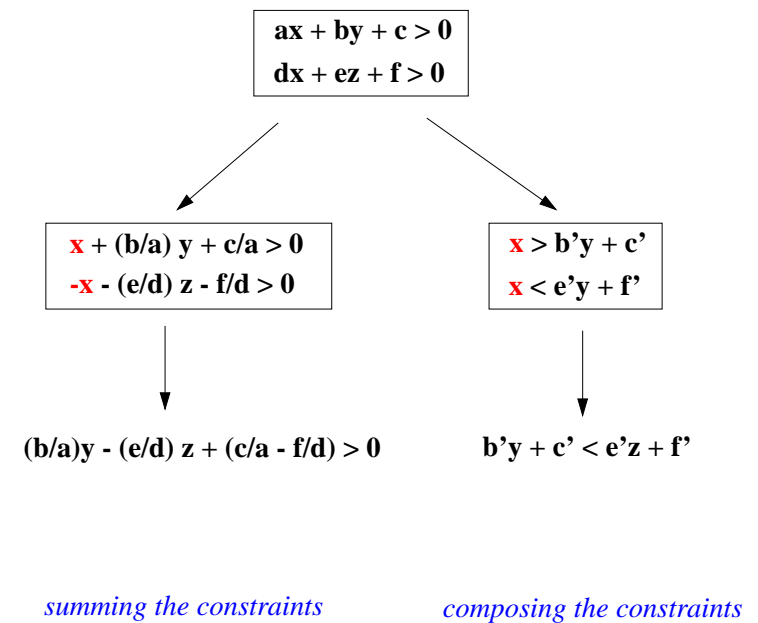
- Given  $m$  constraints over  $k$  variables, the time complexity of variable elimination is  $O(m^2k)$

Previous best time complexity for linear programming is  $O(mk^2 \log m)$  [HN94]

- Given a set  $E$  of 2-variable constraints, the union  $E'$  of projections of  $E$  onto all variable subsets of size 1 or 2 is:

- equivalent to  $E$
- globally consistent

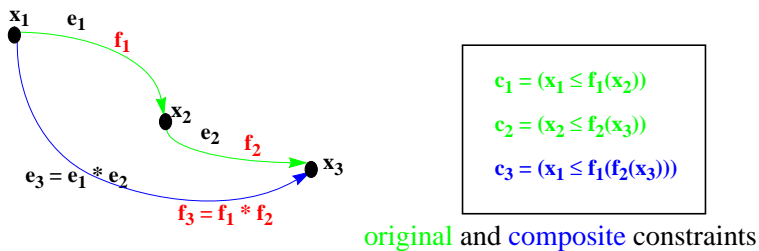
## Variable Elimination



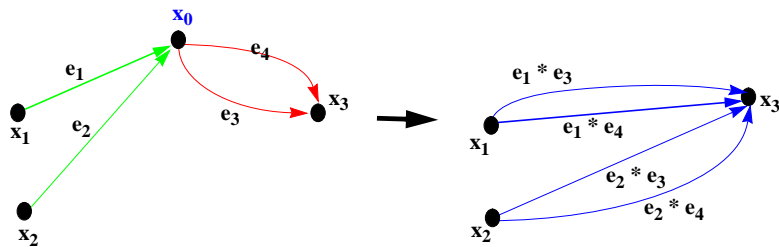
## Variable Elimination for Monotone Constraints

### A variant of the Fourier-Motzkin Algorithm:

1. Represent constraint set by a graph.



2. Use edge composition to do variable elimination.



delete incoming and outgoing edges, add composite edges.  
(j incoming edges, k outgoing edges: jk composite edges)

## Variable Elimination for Monotone Constraints Modifying the Algorithm

1. Represent constraint set by a graph.
2. Assign a **domain** and a **range** to each constraint.
3. Use edge composition to do variable elimination.
4. Use domains and ranges to **prune redundant constraints**.
5. **Count bounding points** of domains and ranges to prove strong polynomiality.

## Variable Elimination for Monotone Constraints Domains and Ranges

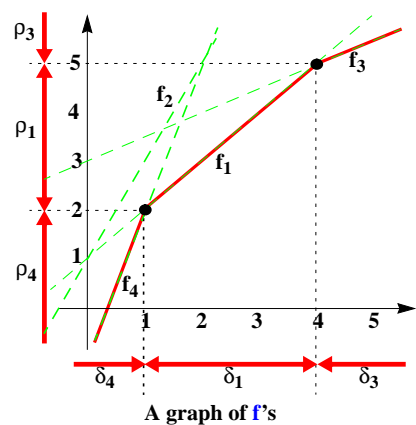
$$c_1 = (x_1 \leq x_2 + 1)$$

$$c_2 = (x_1 \leq 2x_2 + 1)$$

$$c_3 = (x_1 \leq .5x_2 + 3)$$

$$c_4 = (x_1 \leq 3x_2 - 1)$$

Constraints of the form  $x_1 \leq f(x_2)$



Constraint	Domain	Range
$c_4$	$[-\infty, 1]$	$[-\infty, 2]$
$c_1$	$[1, 4]$	$[2, 5]$
$c_3$	$[4, +\infty]$	$[5, +\infty]$
$c$	$\varepsilon$	$\varepsilon$

*A constraint with a trivial domain is redundant.*

## Variable Elimination for Monotone Constraints Observations

- *A constraint with a trivial domain is redundant.*
- *If  $\text{range}(c_1)$  and  $\text{domain}(c_2)$  do not intersect,  $c_1 * c_2$  is redundant.*
- *Out of  $jk$  composite constraints, at most  $(j+k-1)$  are not redundant.*
- *The concepts of range and domain generalize to higher dimensions.*

To make CQLs practical  
it is important  
to solve the indexing problem

## Indexing in CQLs

$X$	$Y$	$Z$
$\alpha_1 \leq x \leq \alpha'_1 \wedge f_1(x, y, z)$		
$\alpha_2 \leq x \leq \alpha'_2 \wedge f_2(x, y, z)$		
$\alpha_3 \leq x \leq \alpha'_3 \wedge f_3(x, y, z)$		

### A Constraint Database

- When generalized tuples represent convex sets, the **projection** of a tuple on  $x$  is an **interval** ( $\alpha \leq x \leq \alpha'$ )
- For **external interval management**, we want **B+ tree** like performance:

**INSERT / DELETE:**  $O(\log_B N)$  I/Os worst-case

**FIND / RANGE:**  $O(\log_B N + k/B)$  I/Os worst-case

**SPACE:**  $O(N/B)$  disk blocks

( $N$  - # of objects in DB,  $B$  - disk block size,  $k$  - # of objects returned)

## Some Indexing Results

*can interval management  
achieve  $B^+$ -tree like performance?*

Using **Metablock Trees** [KKVV PODS'93]:

- **FIND( $X = \alpha$ )**:  $\log_B N + k/B$  I/Os (stabbing query)
- **RANGE( $\alpha' \leq X \leq \alpha''$ )**:  $\log_B N + k/B$  I/Os (interval intersection)
- **INSERT(tuple)**:  $(\log_B N)^2 / B$  amortized (insert interval)
- **SPACE**:  $O(N/B)$  disk blocks
- **DELETE**: cannot be handled by metablock trees

## The next challenge: CQA Optimization

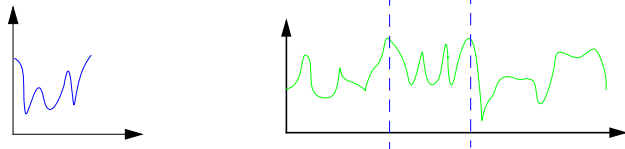
- **Lazy evaluation**
  - for linear and polynomial constraints
  - avoid quantifier elimination for intermediate results
- **Use of implementational information**
  - analogous to System R
  - for estimating expected execution cost
  - heuristics for choosing execution strategy
- **Exploiting expression equivalence**
  - transformation rules between equivalent expressions
  - heuristics for choosing preferable transformations
- **Dynamic view maintenance**
  - for recomputing queries after few updates
  - use previous query result to avoid recomputation

## Time-Series Databases

*temperature readings, heart rate charts, glucose levels, ...*

- **A typical query:**

Has the patient experienced prior  
heart rate fluctuations similar to a given one?



- **Technical issues:**

- sequence shape may be more important than the values
- in real life, time-series sequences almost never match exactly
- there is **lots** of data
- the notion of **similarity** may vary
- quick **response time**

- **What we want to support:**

- flexible constraint-driven intuitive **query specification**
- query mechanism should allow **approximate similarity** searches
- data **indexing** mechanism for fast searches

## Similarity Querying of Time-Series Data (GK'95)

### Semantics of similarity querying:

- **constraint-based definition** of similarity
- **normal form** for each sequence
- **equivalence classes** for similar sequences
- **distance function** between classes

### Syntax of similarity querying:

- **constraint-based** declarative specification
- flexible, **intuitive** syntax

### Index Structure for Spatial Access:

- **no false dismissals**
- **much faster** than linear scan of data
- store a **fingerprint** for each subsequence
- fingerprints are **DFT**-based

## Integration of Constraint Programming and Databases

- **Language Issues**

- the **data complexity** of various CQLs
- combining various **optimization methods** with CQLs
- **complex objects**: what is the interaction of objects and constraints?
- **recursion, aggregation**: how to guarantee safe and efficient querying?

- **Implementation**

- the proper **canonical forms** for various CQLs
- can interval management achieve B<sup>+</sup>-tree like **performance**?

- **Applications**

- **geographical** databases
- **computer-aided design** (product data management)
- **temporal** databases

## Some Current Research CDB implementations

**C3 - GMU**

**DISCO - U.Lincoln/Nebraska**

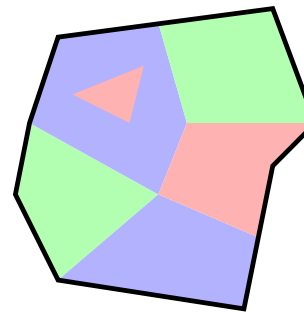
**GIS - INRIA/Roquencourt**

**?**

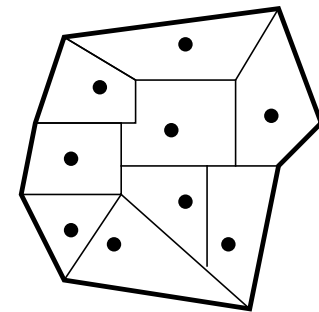
Here went the picture of Paris hiking in Cassis, France  
(see it in the Album on Paris' memorial pages)

**DEDICATED TO PARIS C. KANELLAKIS**

## Spatial Databases - 1



city zoning map



map of fire stations

### Map overlay

- User may want to **query about both maps at once**:
  - Which fire stations serve only the industrial zones?
  - Which residential zones are served by 2 or less fire stations?
- This is a **fundamental operation in spatial databases**.
- **Existing systems perform very poorly on many examples.**  
The reasons range from poor implementation to more fundamental issues of errors in arithmetic operations.

## Spatial Databases - 2

**Constraint DBs** can provide reliable support for map overlay.

- **By mapping overlay problems to linear optimization problems.**

There is reliable technology already available for solving such problems.

- **By using lazy evaluation techniques.**

An implicit representation of the overlay can be stored, with the (trouble-prone) explicit representation computed only when absolutely necessary.

### Example:

- Store the city map data in a **constraint database** with 2 relations:

*ZONES (zone\_no, zone\_type, area\_included)*

*FIRE\_STATIONS (station\_no, station\_locn, area\_served)*

- Perform a **join operation** to access overlay information.