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

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**

  ```
<table>
<thead>
<tr>
<th></th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>e1</td>
<td>d1</td>
</tr>
<tr>
<td>e2</td>
<td>d2</td>
</tr>
<tr>
<td>e3</td>
<td>d4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
  works_in(e, d)  
  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**

  ![Diagram](Diagram)

  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 | (\exists d) works_in(e, d) \land 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), ... \}`

- **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| >> |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

<table>
<thead>
<tr>
<th>Logic Programming</th>
<th>Datalog</th>
<th>CLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>general term</td>
<td>no function symbols</td>
<td>constraints in programs</td>
</tr>
<tr>
<td>unification</td>
<td>efficient bottom-up evaluation semantics</td>
<td>constraint solving</td>
</tr>
<tr>
<td>too costly</td>
<td>cleaner DB theory</td>
<td>replaces unification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>richer data model</td>
</tr>
</tbody>
</table>

- early 80’s
- mid-late 80’s

Datalog - no function symbols
- efficient bottom-up evaluation semantics
- cleaner DB theory
The world is changing...

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

Merging Datalog and CLP

Logic Programming

Datalog

CLP

CQL

- no function symbols
- efficient bottom-up evaluation semantics
- constraints in queries and data
- constraint solving replaces unification
- richer data model and clean DB theory
Rectangle Intersection

a spatiotemporal database example

Relational Model:

- Each rectangle is a tuple
  \[
  \text{Rect}(n, a, b, c, d) \iff n = n_i, a = a_i, b = b_i, c = c_i, d = d_i.
  \]

- Queries contain ad-hoc methods
  \[
  \text{Intsec}(v_1, v_2) \iff \text{Rect}(v_1, \alpha_1, \beta_1, \gamma_1, \delta_1), \text{Rect}(v_2, \alpha_2, \beta_2, \gamma_2, \delta_2).
  \]

  \[
  \text{Intersect}(\alpha_1, \beta_1, \gamma_1, \delta_1, \alpha_2, \beta_2, \gamma_2, \delta_2))}
  \]

  \text{query is sensitive to shape of rectangles}

Constraint model:

- Each point in each rectangle is a tuple (infinitely many).
  We use a finite representation, in the form of constraints.
  \[
  \text{Rect}_pt(\text{Name}, x, y) \iff \text{Name} = n_i, x \leq c_i, y \leq d_i.
  \]

- The semantics of the relation is the (possibly infinite) set of all the tuples \((\text{Name}, x, y)\) satisfying the constraints.

Duality of representation and meaning

<table>
<thead>
<tr>
<th>Name</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_1)</td>
<td>(x_1)</td>
<td>(y_1)</td>
</tr>
<tr>
<td>(n_1)</td>
<td>(x_2)</td>
<td>(y_2)</td>
</tr>
<tr>
<td>(n_1)</td>
<td>(x_3)</td>
<td>(y_3)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(n_2)</td>
<td>(x'_1)</td>
<td>(y'_1)</td>
</tr>
<tr>
<td>(n_2)</td>
<td>(x'_2)</td>
<td>(y'_2)</td>
</tr>
</tbody>
</table>

Constraint representation of \(R\)

- Report intersections by the query:
  \[
  \text{Intsec}(v_1, v_2) \iff \text{Rect}_pt(v_1, x, y), \text{Rect}_pt(v_2, x, y)
  \]

  \text{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

<table>
<thead>
<tr>
<th></th>
<th>Relational Calculus</th>
<th>(Linear) Datalog with negation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>$\mathsf{AC}^0$</td>
<td>(NC) PTIME</td>
</tr>
<tr>
<td>Dense-order, Temporal</td>
<td>$\mathsf{AC}^0$</td>
<td>(NC) PTIME</td>
</tr>
<tr>
<td>Real Linear</td>
<td>LOGSPACE (?)</td>
<td>-</td>
</tr>
<tr>
<td>Real Polynomial</td>
<td>NC (?)</td>
<td>-</td>
</tr>
<tr>
<td>Integer Order</td>
<td>PTIME (?)</td>
<td>-</td>
</tr>
<tr>
<td>Integer Linear</td>
<td>PTIME (??)</td>
<td>-</td>
</tr>
</tbody>
</table>

### Various Constraint Classes

<table>
<thead>
<tr>
<th>Constraint Class</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>$x = 3, y = 4$</td>
</tr>
<tr>
<td>Dense-order</td>
<td>$x &gt; 2, y \leq 3, y &lt; x$</td>
</tr>
<tr>
<td>Temporal</td>
<td>$y &lt; x + 7$</td>
</tr>
<tr>
<td>Monotone</td>
<td>$y &lt; ax + 7, a &gt; 0$</td>
</tr>
<tr>
<td>Linear</td>
<td>$5y + 3x &lt; 7, y - 2x &gt; 9$</td>
</tr>
<tr>
<td>Polynomial</td>
<td>$x^2 + y^2 = 4, x^2 &lt; 3$</td>
</tr>
</tbody>
</table>
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

- **High-level query specification** (KKR’90)
- **Algebraic expressions** (procedural query specification) (GK’96)
- **Data access methods** (KKVV’94)
- **Query optimization**
- **Applications** (GK’95)
### 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
  \[ \Pi_{e,m} (\text{works} \in > \text{dept}_\text{mgr}) \]
  - 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^0** 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

  \[ r_1(e, d) = \left( \text{works}_\text{in}(e, d) \land d = \text{CS} \right) \quad r_2(e) = r_1(e, d) \]

- Interpreting CQA operators

  \[ R(x, y, z) \]

  \[
  \begin{array}{ccc}
  (0 < x < y < 0.5, z = 0.5) & (0 < x < y < 0.5, z = 0.5) & (0 < x < y < 1) \\
  (1, 1, 1) & (1, 1, 0) & (1, 1, 0)
  \end{array}
  \]

  - **select** \(_{(z=0.5)}R\)
  - **project** \(_{x,y}R\)

**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

Preserving the relational algebra syntax and bottom-up semantics.
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

\[
\begin{align*}
  ax + by + c & > 0 \\
  dx + ez + f & > 0 \\
  x + (b/a)y + c/a & > 0 \\
  (b/a)y - (e/d)z + (c/a - f/d) & > 0 \\
  x > b'y + c' \\
  x < e'y + f' \\
  b'y + c' & < e'z + f'
\end{align*}
\]

summing the constraints

composing the constraints
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.

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

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

\[
\begin{align*}
c_1 &= (x_1 \leq x_2 + 1) \\
c_2 &= (x_1 \leq 2x_2 + 1) \\
c_3 &= (x_1 \leq 5x_2 + 3) \\
c_4 &= (x_1 \leq 3x_2 - 1)
\end{align*}
\]

Domains and ranges

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, \( c_1 \ast c_2 \) 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.

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:

<table>
<thead>
<tr>
<th>Insert/Delete</th>
<th>$O(\log_B N)$ I/Os worst-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find/Range</td>
<td>$O(\log_B N + k/B)$ I/Os worst-case</td>
</tr>
<tr>
<td>Space</td>
<td>$O(N/B)$ disk blocks</td>
</tr>
</tbody>
</table>

($N$ - # 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^4-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)

Spatial Databases - 1

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.

**city zoning map**

**map of fire stations**
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.