Schema Mappings and Data Exchange

Lecture #2

EASSLC 2012

Southwest University

August 2012

The Relational Data Model (E.F. Codd – 1970)

- The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.
- Question: What is a relation?
- Answer:
  - Formally, a relation is a subset of a cartesian product of sets.
  - Informally, a relation is a "table" with rows and columns.

CHECKING Table

branch-name	**account-no**	**customer-name**	**balance**
Orsay	10991-06284	Abiteboul	$3,567.53
Hawthorne	10992-35671	Hull	$11,245.75
…	…	…	…

Relational Structures vs. Relational Databases

- Relational Structure

  $\mathbf{A} = (A, R_1,\ldots,R_m)$

  - A is the **universe** of **A**
  - $R_1,\ldots,R_m$ are the relations of **A**

- Relational Database

  $\mathbf{D} = (R_1,\ldots,R_m)$

- Thus, a relational database can be thought of as a relational structure **without** its universe.
  - And this causes some problems down the road …

Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

- Relational Algebra, which is a procedural language.
  - It is an algebraic formalism in which queries are expressed by applying a sequence of operations to relations.

- Relational Calculus, which is a declarative language.
  - It is a logical formalism in which queries are expressed as formulas of first-order logic.

Codd's Theorem: Relational Algebra and Relational Calculus are "essentially equivalent" in terms of expressive power.

(but what does this really mean?)

The Five Basic Operations of Relational Algebra

- Group I: Three standard set-theoretic binary operations:
  - Union
  - Difference
  - Cartesian Product.
- Group II. Two special unary operations on relations:
  - Projection
  - Selection.
- Relational Algebra consists of all expressions obtained by combining these five basic operations in syntactically correct ways.

Relational Algebra

- Definition: A relational algebra expression is a string obtained from relation schemas using union, difference, cartesian product, projection, and selection.

- Context-free grammar for relational algebra expressions:

  E := R, S, … | $(E_1 \cup E_2)$ | $(E_1 - E_2)$ | $(E_1 \times E_2)$ | $\pi_L(E)$ | $\sigma_\Theta(E)$,

  where
  - R, S, … are relation schemas
  - L is a list of attributes
  - Θ is a condition.

Natural Join

Given TEACHES(fac-name,course,term) and
ENROLLS(stud-name, course,term):
To compute TAUGHT-BY(stud-name,course,term,fac-name)

1. ENROLLS $\times$ TEACHES
2. $\sigma_{\text{T.course = E.course} \wedge \text{T.term = E.term}}$ (ENROLLS $\times$ TEACHES)
3. $\pi_{\text{stud-name,E.course,E.term,fac-name}}$
   ($\sigma_{\text{T.course = E.course} \wedge \text{T.term = E.term}}$ (ENROLLS $\times$ TEACHES))

The result is ENROLLS $\bowtie$ TEACHES.

SQL vs. Relational Algebra

SQL	**Relational Algebra**
SELECT	Projection π
FROM	Cartesian Product $\times$
WHERE	Selection σ

Semantics of SQL via interpretation to Relational Algebra

SELECT R_{i1}.A1, …, R_{im}.A.m
FROM R_1, …,R_K
WHERE Ψ

$$= \pi_{Ri1.A1, \ldots, Rim.A.m} (\sigma_{\Psi} (R_1 \times \ldots \times R_K))$$

Relational Calculus

- In addition to relational algebra, Codd introduced relational calculus.
- Relational calculus is a declarative database query language based on first-order logic.
- Relational calculus comes into two different flavors:
  - Tuple relational calculus
  - Domain relational calculus.

  We will focus on domain relational calculus.

  There is an easy translation between these two formalisms.
- Codd's main technical result is that relational algebra and relational calculus have "essentially" the same expressive power.

Relational Calculus (First-Order Logic for Databases)

- First-order variables: x, y, z, …, x_1, …,x_k,…
  - They range over values that may occur in tables.
- Relation symbols: R, S, T, … of specified arities (names of relations)
- Atomic (Basic) Formulas:
  - $R(x_1,\ldots,x_k)$, where R is a k-ary relation symbol (alternatively, $(x_1,\ldots,x_k) \in R$; the variables need not be distinct)
  - (x op y), where op is one of =, ≠, <, >, ≤, ≥
  - (x op c), where c is a constant and op is one of =, ≠, <, >, ≤, ≥.
- Relational Calculus Formulas:
  - Every atomic formula is a relational calculus formula.
  - If φ and ψ are relational calculus formulas, then so are:
    - $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\neg \psi$, $(\varphi \rightarrow \psi)$ (propositional connectives)
    - $(\exists x\ \varphi)$ (existential quantification)
    - $(\forall x\ \varphi)$ (universal quantification).

Relational Calculus as a Database Query Language

Definition:

- A relational calculus expression is an expression of the form

$$\{ (x_1,\ldots,x_k):\ \varphi(x_1,\ldots x_k) \},$$

  where $\varphi(x_1,\ldots,x_k)$ is a relational calculus formula with $x_1,\ldots,x_k$ as its free variables.
- When applied to a relational database I, this relational calculus expression returns the k-ary relation that consists of all k-tuples $(a_1,\ldots,a_k)$ that make the formula "true" on I.

Example: The relational calculus expression

$$\{ (x,y):\ \exists z(E(x,z) \wedge E(z,y)) \}$$

returns the set P of all pairs of nodes (a,b) that are connected via a path of length 2.

Natural Join in Relational Calculus

Given TEACHES(fac-name,course,term) and
ENROLLS(stud-name, course,term):

Compute TAUGHT-BY(stud-name,course,term,fac-name)

In relational algebra:

$$\pi_{\text{stud-name,E.course,E.term,fac-name}} (\sigma_{\text{T.course = E.course} \wedge \text{T.term = E.term}} (\text{ENROLLS} \times \text{TEACHES}))$$

In relational calculus:

$$\{ (s,c,t,f):\ \text{ENROLL}(s,c,t) \wedge \text{TEACHES}(f,c,t) \}$$

Relational Algebra vs. Relational Calculus

Codd's Theorem (informal statement):
Relational Algebra and Relational Calculus have "essentially" the same expressive power, i.e., they can express the same queries.

Note: It is **not** true that for every relational calculus expression φ, there is an equivalent relational algebra expression E.

Examples:

- $\{ (x_1,\ldots,x_k): \neg R(x_1,\ldots,x_k) \}$
- $\{ x: \forall y,z \text{ ENROLLS}(x,y,z) \}$, where ENROLLS(s-name,course,term)

From Relational Calculus to Relational Algebra

Note: The previous relational calculus expression may produce different answers when we consider different domains over which the variables are interpreted.

Example: If the variables $x_1,\ldots,x_k$ range over a domain D, then

$$\{(x_1,\ldots,x_k): \neg R(x_1,\ldots,x_k)\} = D^k - R.$$

Fact:

- The relational calculus expression $\{ (x_1,\ldots,x_k): \neg R(x_1,\ldots,x_k) \}$ is **not** "domain independent".
- The relational calculus expression $\{(x_1,\ldots,x_k): S(x_1,..,x_k) \wedge \neg R(x_1,\ldots,x_k)\}$ is "domain independent".

Active Domain and Active Domain Interpretation

Definition:

- The active domain adom(I) of a relational database instance I is the set of all values that occur in the relations of I.

- Let $\varphi(x_1,\ldots,x_k)$ be a relational calculus formula and let I be a relational database instance. Then

$$\varphi^{adom}(I)$$

is the result of evaluating $\varphi(x_1,\ldots,x_k)$ over adom(I) and I, that is,

  - all variables and quantifiers are assumed to range over adom(I);
  - the relation symbols in φ are interpreted by the relations in I.

Queries

Definition: Let **S** be a relational database schema.

A k-ary query on **S** is a function q defined on database instances over S such that if I is a database instance over S, then q(I) is a k-ary relation that is invariant under isomorphisms and has values among those occurring in the relations in I

(i.e., if h: I $\rightarrow$ J is an isomorphism, then q(J) = h(q(I)).

Note:

- All "queries" that we have expressed in relational algebra and/or in relational calculus so far are queries in the above formal sense.
- In particular, a relational calculus expression of the form

  $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots x_k)\}$ defines a k-ary query.

Equivalence of Relational Algebra and Relational Calculus

Theorem: The following are equivalent for a k-ary query q:

1. There is a relational algebra expression E such that q(I) = E(I), for every database instance I
   (in other words, q is expressible in relational algebra).

2. There is a relational calculus formula ψ such that q(I) = ψ^{adom} (I)
   (in other words, q is expressible in relational calculus under the active domain interpretation).

From Relational Algebra to Relational Calculus

(1) $\Rightarrow$ (2) For every relational expression E, there is an equivalent relational calculus expression $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots x_k)\}$.

Proof: By induction on the construction of rel. algebra expressions.

- If E is a relation R of arity k, then we take $\{(x_1,\ldots,x_k): E(x_1,\ldots,x_k)\}$.
- Assume E_1 and E_2 are expressible by $\{(x_1,\ldots,x_k): \varphi_1(x_1,\ldots,x_k)\}$ and by $\{(x_1,\ldots,x_m): \varphi_2(x_1,\ldots,x_m)\}$. Then
  - $E_1 \cup E_2$ is expressible by $\{(x_1,\ldots,x_k): \varphi_1(x_1,\ldots,x_k) \vee \varphi_2(x_1,\ldots,x_k)\}$.
  - $E_1 - E_2$ is expressible by $\{(x_1,\ldots,x_k): \varphi_1(x_1,\ldots,x_k) \wedge \neg\varphi_2(x_1,\ldots,x_k)\}$.
  - $E_1 \times E_2$ is expressible by $\{(x_1,\ldots,x_k,y_1,\ldots,y_m): \varphi_1 (x_1,\ldots,x_k) \wedge \varphi_2(y_1,\ldots,y_m)\}$

From Relational Algebra to Relational Calculus

(1) $\Rightarrow$ (2) For every relational expression E, there is an equivalent relational calculus expression $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots x_k)\}$.

Proof: (continued)

- Assume that E is expressible by $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots,x_k)\}$.
  Then
  - $\pi_{1,3}(E)$ is expressible by
    $\{(x_1,x_3): (\exists\, x_2)(\exists\, x_4)\ \ldots(\exists\, x_k)\ \varphi(x_1,\ldots,x_k)\ \}$
  - $\sigma_{\Theta}(E)$ is expressible by
    $\{(x_1,\ldots,x_k): \Theta^* \wedge \varphi(x_1,\ldots,x_k)\}$, where Θ^* is the rewriting of Θ as a formula of relational calculus.

Equivalence of Relational Algebra and Relational Calculus

Proof (Sketch):

2. $\Rightarrow$ 1.

- Show first that for every relational database schema **S**, there is a relational algebra expression E such that for every database instance I, we have that adom(I) = E(I).

- Use the above fact and induction on the construction of relational calculus formulas to obtain a translation of relational calculus under the active domain interpretation to relational algebra.

Equivalence of Relational Algebra and Relational Calculus

- In this translation, the most interesting part is the simulation of the universal quantifier $\forall$ in relational algebra.
  - It uses the logical equivalence $\forall y\psi \equiv \neg\exists y\neg\psi$
- As an illustration, consider $\forall y R(x,y)$.
  - $\forall y R(x,y) \equiv \neg\exists y\neg R(x,y)$
  - adom(I) = $\pi_1(R) \cup \pi_2(R)$

Rel.Calc. formula φ	Relational Algebra Expression for φ^{adom}
$\neg$ R(x,y)	$(\pi_1(R) \cup \pi_2(R))\times(\pi_1(R) \cup \pi_2(R)) - R$
$\exists y\neg R(x,y)$	$\pi_1((\pi_1(R) \cup \pi_2(R))\times(\pi_1(R) \cup \pi_2(R)) - R)$
$\neg\exists y\neg R(x,y)$	$(\pi_1(R) \cup \pi_2(R)) - (\pi_1((\pi_1(R) \cup \pi_2(R))\times(\pi_1(R) \cup \pi_2(R)) - R))$

Equivalence of Relational Algebra and Relational Calculus

Remarks:

- The Equivalence Theorem is effective. Specifically, the proof of this theorem yields two algorithms:
  - an algorithm for translating from relational algebra to domain independent relational calculus, and
  - an algorithm from translating from domain independent relational calculus to relational algebra.
- Each of these two algorithms runs in linear time.

Queries

Definition: Let **S** be a relational database schema.

- A k-ary query on **S** is a function q defined on database instances over S such that if I is a database instance over S, then q(I) is a k-ary relation on adom(I) that is invariant under isomorphisms (i.e., if h: $I \rightarrow J$ is an isomorphism, then q(J) = h(q(I)).

- A Boolean query on **S** is a function q defined on database instances over S such that if I is a database instance over S, then q(I) = 0 or q(I) = 1, and q(I) is invariant under isomorphisms.

Example: The following are Boolean queries on graphs:

- Given a graph E (binary relation), is the diameter of E at most 3?
- Given a graph E (binary relation), is E connected?

Three Fundamental Algorithmic Problems about Queries

- The Query Evaluation Problem: Given a query q and a database instance I, find q(I).

- The Query Equivalence Problem: Given two queries q and q' of the same arity, is it the case that $q \equiv q'$?

  (i.e., is it the case that, for every database instance I, we have that q(I) = q'(I)?)

- The Query Containment Problem: Given two queries q and q' of the same arity, is it the case that $q \subseteq q'$?

  (i.e., is it the case that, for every database instance I, we have that $q(I) \subseteq q'(I)$?)

Three Fundamental Algorithmic Problems about Queries

- The Query Evaluation Problem is the main problem in query processing.
- The Query Equivalence Problem underlies query processing and optimization, as we often need to transform a given query to an equivalent one.
- The Query Containment Problem and Query Equivalence Problem are closely related to each other:
  - $q \equiv q'$ if and only if $q \subseteq q'$ and $q' \subseteq q$.
  - $q \subseteq q'$ if and only if $q \equiv (q \wedge q')$.

Three Fundamental Algorithmic Problems about Queries

- Our goal is to investigate the algorithmic aspects of these problems for queries expressible in relational algebra/relational calculus.
- The questions we want to address are:
  - How can we measure the precise "difficulty" of these problems?
  - Are there "good" algorithms for solving these problems?
  - If not, are there special cases of these problems for which "good" algorithms exist?

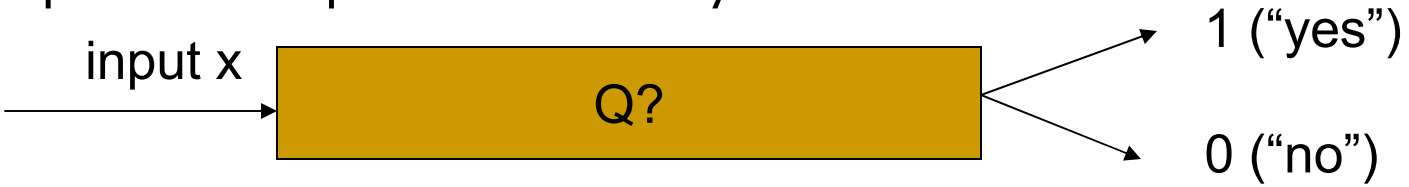
Three Fundamental Algorithmic Problems about Queries

Our study of these problems will use concepts and methods from two different, yet related, areas:

- Mathematical Logic:
  - Computability Theory and Undecidable Problems
- Computational Complexity Theory:
  - Complexity Classes and Complete Problems
  - In particular, the classes P and NP, and NP-complete problems.

Decision Problems and Languages

- Definition (informal): A decision problem Q consists of a set of inputs and a question with a "yes" or "no" answer for each input.



- Definition: A decision problem is
  - Decidable if there is an algorithm for solving it;
  - Undecidable if there is no algorithm for solving it.

Undecidable Problems

Theorem: The following problems are undecidable:

- The Halting Problem (A. Turing – 1936): Given a Turing machine M and an input x, does M halt on x?

- The Finite Validity Problem (B. Trakhtenbrot – 1949): Given a first-order formula φ on graphs, is φ true on every finite graph?

Undecidable Problems

- The Finite Validity Problem (B. Trakhtenbrot – 1949): Given a first-order sentence φ on graphs, is φ true on every finite graph?
- Examples of Finitely Valid Formulas:
  - $\forall x(E(x,x) \rightarrow \exists y E(x,y))$
  - $\forall x \forall y(E(x,x) \wedge x = y \rightarrow E(y,y))$
  - "if E is a total order, then E has a biggest element"
- Example of Non-Finitely Valid Formulas:
  - $\forall x \forall y (E(x,y) \rightarrow E(y,x))$
  - $(\forall x \exists y E(x,y)) \rightarrow (\exists y \forall x E(x,y))$
- The undecidability of the Finite Validity Problem means that there is **no** algorithm for telling formulas in the first group from formulas in the second group.

The Reduction Method

- By now there is a vast library of undecidable problems.
- The Reduction Method is the main technique for establishing undecidability.
- Reduction Method: To show that a problem L* is undecidable, it suffices to find an undecidable problem L and a computable function f such that for every input x, we have that

$$x \in L \quad \Leftrightarrow \quad f(x) \in L^*.$$

  - Such a function f is called a reduction of L to L*
  - $L \preccurlyeq L^*$ means that there is a reduction of L to L*.

The Reduction Method

- The Halting Problem was the first fundamental decision problem shown to be undecidable.

- The Finite Validity Problem was shown to be undecidable by showing that Halting Problem $\preccurlyeq$ Finite Validity Problem.

- Many database problems have been shown to be undecidable via reductions from
  - The Halting Problem

  or
  - The Finite Validity Problem

Undecidability of The Query Equivalence Problem

- The Query Equivalence Problem: Given two queries q and q' of the same arity, is it the case that $q \equiv q'$?

  (i.e., is q(I) = q'(I) on every database instance I?)

- Theorem: The Query Equivalence Problem for relational calculus queries is undecidable.

  Proof: Finite Validity Problem $\preccurlyeq$ Query Equivalence Problem

  - To see, this let ψ^* be a fixed finitely valid relational calculus sentence (say, $\forall$ x(E(x,x) $\rightarrow \exists$ yE(x,y))).
  - Then, for every relational calculus sentence φ, we have that

    φ is finitely valid $\Leftrightarrow$ $\varphi \equiv \psi^*$.

Undecidability of the Query Containment Problem

- The Query Containment Problem: Given two queries q and q′ of the same arity, is it the case that $q \subseteq q'$?

  (i.e., is $q(I) \subseteq q'(I)$ on every database instance I?)

- Corollary: The Query Containment Problem for relational calculus queries in undecidable.

  Proof: Query Equivalence $\preccurlyeq$ Query Containment, since

  $$q \equiv q' \Leftrightarrow q \subseteq q' \text{ and } q' \subseteq q.$$

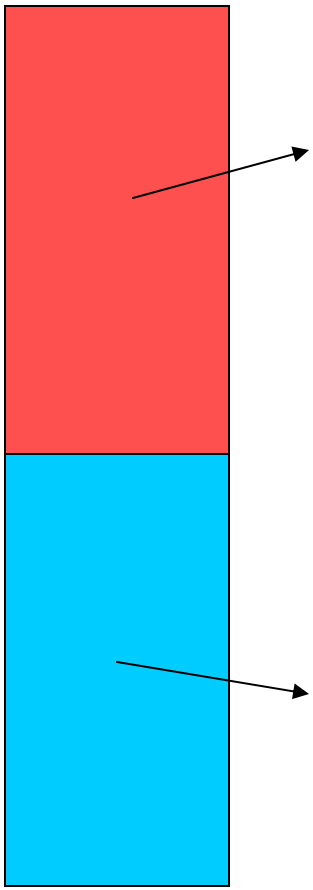
- Notice the chain of reductions:

  Halting Problem $\preccurlyeq$ Finite Validity $\preccurlyeq$ Query Equiv. $\preccurlyeq$ Query Cont.

The Query Evaluation Problem

- The Query Evaluation Problem: Given a query q and a database instance I, find q(I).
- The Query Evaluation Problem for relational calculus queries is decidable, but, as we will see, it has high computational complexity.
- To understand the precise algorithmic difficulty of the Query Evaluation Problem, we need some basic notions and results from computational complexity.

Decidable Problems and Computational Complexity



- Computational Complexity is the quantitative study of decidable problems.
- "From these and other considerations grew our deep conviction that **there must be quantitative laws that govern the behavior of information and computing**. The results of this research effort were summarized in our first paper on this topic, which also named this new research area, "On the computational complexity of algorithms"."

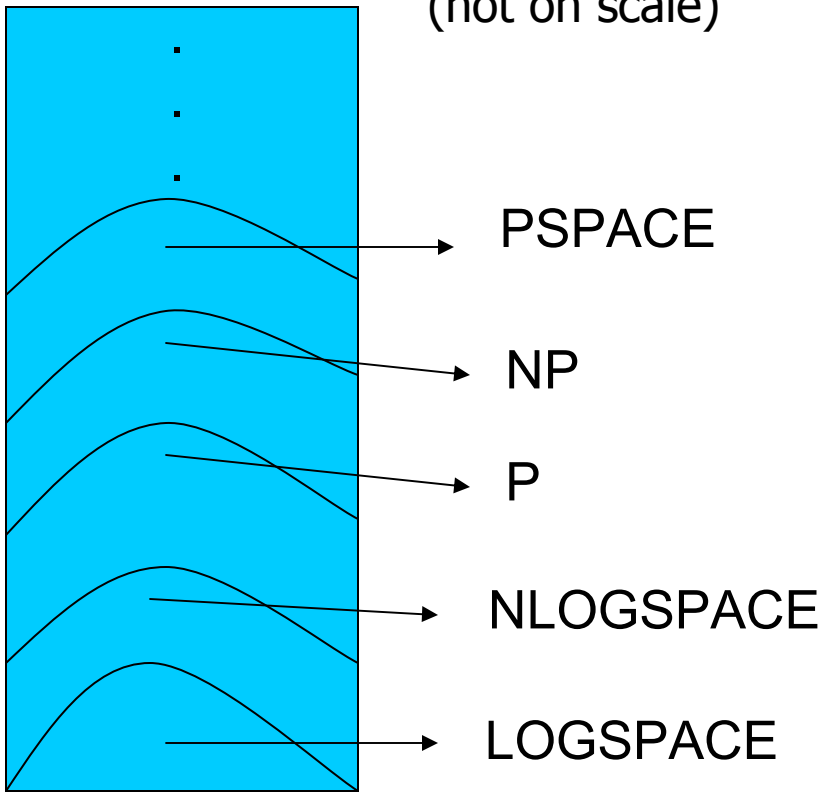
J. Hartmanis, Turing Award Lecture, 1993

Computational Complexity Classes

- Decidable problems are grouped together in computational complexity classes.
- Each computational complexity class consists of all problems that can be solved in a computational model under certain restrictions on the resources used to solve the problem.
- Examples of computational models:
  - Turing Machine TM (deterministic Turing machine)
  - Non-deterministic Turing machine NTM
  - …
- Examples of resources:
  - Amount of time needed to solve the problem
  - Amount of space (memory) needed to solve the problem.
  - …

Computational Complexity Classes

Classification of Decidable Problems
(not on scale)



There are many other complexity classes. For a comprehensive catalog, visit the Complexity Zoo at

qwiki.stanford.edu/wiki/Complexity_Zoo

Complexity of the Query Evaluation Problem

The Query Evaluation Problem for Relational Calculus:

Given a relational calculus formula φ and a database instance I, find $\varphi^{adom}(I)$.

Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

Proof: We need to show that

- This problem is in PSPACE.
- This problem is PSPACE-hard.

We start with the second task.

Complexity of the Query Evaluation Problem

- Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-hard.
- Proof: Show that

  Quantified Boolean Formulas $\preccurlyeq_p$ Query Evaluation for Rel. Calc.

  Given QBF $\forall x_1 \exists x_2 \ldots \forall x_k \psi$
  - Let V and P be two unary relation symbols
  - Obtain ψ^* from ψ by replacing x_i by $P(x_i)$, and $\neg x_i$ by $\neg P(x_i)$
  - Let I be the database instance with V = {0,1}, P={1}.
  - Then the following statements are equivalent:
    - $\forall x_1 \exists x_2 \ldots \forall x_k \psi$ is true
    - $\forall x_1 (V(x_1) \rightarrow \exists x_2 (V(x_2) \wedge (\ldots \forall x_k (V(x_k) \rightarrow \psi^*)) \ldots)$ is true on I.

Complexity of the Query Evaluation Problem

- Theorem: The Query Evaluation Problem for Relational Calculus is in PSPACE.

  Proof (Hint): Let φ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let I be a database instance.

  - Exponential Time Algorithm: We can find $\varphi^{adom}(I)$, by exhaustively cycling over all possible interpretations of the x_i's.

    This runs in time $O(n^m)$, where $n = |I|$ (size of I).
  - A more careful analysis shows that this algorithm can be implemented in $O(m \cdot \log n)$-space.
    - Use m blocks of memory, each holding one of the n elements of adom(I) written in binary (so O(logn) space is used in each block).
    - Maintain also m counters in binary to keep track of the number of elements examined.

$\forall x_1$	$\exists x_2$	…	$\forall x_m$
a_1 in adom(I) written in binary	a_2 in adom(I) written in binary	…	a_m in adom(I) written in binary

Summary

- Relational Algebra and Relational Calculus have "essentially" the same expressive power.
- The Query Equivalence Problem for Relational Calculus in undecidable.
- The Query Containment Problem for Relational Calculus is undecidable.
- The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

Sublanguages of Relational Calculus

- Question: Are there interesting sublanguages of relational calculus for which the Query Containment Problem and the Query Evaluation Problem are "easier" than the full relational calculus?

- Answer:
  - Yes, the language of conjunctive queries is such a sublanguage.

  - Moreover, conjunctive queries are the most frequently asked queries against relational databases.

Conjunctive Queries

- Definition: A conjunctive query is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

  $$\{ (x_1,\ldots,x_k): \exists z_1 \ldots \exists z_m \chi(x_1, \ldots,x_k, z_1,\ldots,z_k) \},$$

  where $\chi(x_1, \ldots,x_k, z_1,\ldots,z_k)$ is a conjunction of atomic formulas of the form $R(y_1,\ldots,y_m)$.
  - Equivalently, a conjunctive query is a query expressible by a relational algebra expression of the form

    $\pi_X(\sigma_\Theta(R_1 \times \ldots \times R_n))$, where

    Θ is a conjunction of equality atomic formulas (equijoin).
  - Equivalently, a conjunctive query is a query expressible by an SQL expression of the form

    SELECT <list of attributes>

    FROM <list of relation names>

    WHERE <conjunction of equalities>

Conjunctive Queries

- Definition: A conjunctive query is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

  $$\{ (x_1,\ldots,x_k): \exists z_1 \ldots \exists z_m \chi(x_1, \ldots,x_k, z_1,\ldots,z_k) \}$$

  - A conjunctive query can be written as a logic-programming rule:

    $$Q(x_1,\ldots,x_k) \text{ :-- } R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n), \text{ where}$$

    - Each variable x_i occurs in the right-hand side of the rule.
    - Each $\mathbf{u}_i$ is a tuple of variables (not necessarily distinct)
    - The variables occurring in the right-hand side (the body), but not in the left-hand side (the head) of the rule are existentially quantified (but the quantifiers are not displayed).
    - "," stands for conjunction.

Conjunctive Queries

Examples:

- Path of Length 2: (Binary query)
  $\{(x,y): \exists\, z\, (E(x,z) \wedge E(z,y))\}$
  - As a relational algebra expression,
    $\pi_{1,4}(\sigma_{\$2 = \$3}\, (E \times E))$
  - As a rule:
    q(x,y) :-- E(x,z), E(z,y)
- Cycle of Length 3: (Boolean query)
  $\exists\, x \exists\, y \exists\, z(E(x,y) \wedge E(y,z) \wedge E(z,x))$
  - As a rule (the head has no variables)
    - Q :-- E(x,z), E(z,y), E(z,x)

Conjunctive Queries

- Every relational join is a conjunctive query:
  P(A,B,C), R(B,C,D) two relation symbols
  - $P \bowtie R = \{(x,y,z,w): P(x,y,z) \wedge R(y,z,w)\}$
  - q(x,y,z,w) :-- P(x,y,z), R(y,z,w)
    (no variables are existentially quantified)
  - SELECT P.A, P.B, P.C, R.D
    FROM P, R
    WHERE P.B = R.B AND P.C = R.C
- Conjunctive queries are also known as SPJ-queries
  (SELECT-PROJECT-JOIN queries)

Conjunctive Query Evaluation and Containment

- Definition: Two fundamental problems about CQs
  - Conjunctive Query Evaluation (CQE):
    Given a conjunctive query q and an instance I, find q(I).
  - Conjunctive Query Containment (CQC):
    - Given two k-ary conjunctive queries q_1 and q_2,
      is it true that $q_1 \subseteq q_2$?
      (i.e., for every instance I, we have that $q_1(I) \subseteq q_2(I)$)
    - Given two Boolean conjunctive queries $q_1$and q_2, is it true that $q_1 \models q_2$? (that is, for all I, if $I \models q_1$, then $I \models q_2$)?
      CQC is logical implication.

CQE vs. CQC

- Recall that for relational calculus queries:
  - The Query Evaluation Problem is PSPACE-complete (combined complexity).
  - The Query Containment Problem is undecidable.

- Theorem: Chandra & Merlin, 1977
  - CQE and CQC are the "same" problem.
  - Moreover, each is an NP-complete problem.

- Question: What is the common link?
- Answer: The Homomorphism Problem