Logic and Databases

Phokion G. Kolaitis

UC Santa Cruz & IBM Research - Almaden
Logic and Databases are inextricably intertwined.

C.J. Date -- 2007
Logic and Databases

• Extensive interaction between logic and databases during the past 45 years.

• Logic provides both a unifying framework and a set of tools for formalizing and studying data management tasks.

• The interaction between logic and databases is a prime example of
  – Logic in Computer Science
    but also
  – Logic from Computer Science
Logic and Databases

Two main uses of logic in databases:

• Logic is used as a database query language to express questions asked against databases.

• Logic is used as a specification language to express integrity constraints in databases.

We will discuss both of these uses with emphasis on the first.
Thematic Roadmap

• Logic and Database Query Languages
  – Relational Algebra and Relational Calculus
  – Conjunctive queries and their variants
  – Datalog

• Query Evaluation, Query Containment, Query Equivalence
  – Decidability and Complexity

• Other Aspects of Conjunctive Query Evaluation

• Alternative Semantics of Queries
  – Bag Databases: Semantics and Conjunctive Query Containment
  – Probabilistic Databases: Semantics and Dichotomy Theorems for Conjunctive Query Evaluation
  – Inconsistent Databases: Semantics and Dichotomy Theorems

• Guest Lecture on Data Provenance by Val Tannen
Relational Databases: How it all got started

• The history of relational databases is the history of a scientific and technological revolution.

• The scientific revolution started in 1970 by Edgar (Ted) F. Codd at the IBM San Jose Research Laboratory (now the IBM Almaden Research Center)

• Codd introduced the relational data model and two database query languages: relational algebra and relational calculus.

Edgar F. Codd, 1923-2003
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

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-no</th>
<th>customer-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orsay</td>
<td>10991-06284</td>
<td>Abiteboul</td>
<td>$13,567.53</td>
</tr>
<tr>
<td>Hawthorne</td>
<td>10992-35671</td>
<td>Hull</td>
<td>$21,245.75</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Relational Database Schemas

• A k-ary relation schema \( R(A_1, A_2, \ldots, A_K) \) is a set \{A_1, A_2, \ldots, A_k\} of k attributes.

  **CHECKING** (branch-name, account-no, customer-name, balance)
  
  • Thus, a k-ary relation schema is a “blueprint” for k-ary relations.
  
  • It is a k-ary relation symbol in logic with names for the positions.

• An instance of a relation schema is a relation conforming to the schema (arities match; also, in DBMS, data types of attributes match).

• A relational database schema is a set of relation schemas \( R_i(A_1, A_2, \ldots, A_{k_i}) \), for \( 1 \leq i \leq m \).

• A relational database instance of a relational schema is a set of relations \( R_i \) each of which is an instance of the relation schema \( R_i \), \( 1 \leq i \leq m \).
Relational Structures vs. Relational Databases

- Relational Structure
  \[ A = (A, R_1, \ldots, R_m) \]
  - A is the **universe** of A
  - \( R_1, \ldots, R_m \) are the relations of A

- Relational Database
  \[ D = (R_1, \ldots, R_m) \]

- 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.
More on the Syntax of the Projection Operation

- **Projection Operation:**
  - **Syntax:** $\pi_{i_1,\ldots,i_m}(R)$, where $R$ is of arity $k$, and $i_1, \ldots, i_m$ are distinct integers from 1 up to $k$.
  - **Semantics:**
    $$\pi_{i_1,\ldots,i_m}(R) = \{(a_1,\ldots,a_m): \text{there is a tuple } (b_1,\ldots,b_k) \text{ in } R \text{ such that } a_1 = b_{i_1}, \ldots, a_m = b_{i_m}\}$$

- **Example:** If $R$ is $R(A,B,C,D)$, then
  $$\pi_{3,1}(R) = \{(c,a): \text{there are } b,d \text{ such that } (a,b,c,d) \in R\} = \pi_{C,A}(R)$$
The Selection Operation

• **Selection** is a family of unary operations of the form
  \[ \sigma_\Theta (R), \]
  where \( R \) is a relation and \( \Theta \) is a *condition* that can be applied as a test to each row of \( R \).

• When a selection operation is applied to \( R \), it returns the subset of \( R \) consisting of all rows that satisfy the condition \( \Theta \).

• A *condition* in the selection operation is an expression built up from:
  – Comparison operators \( =, <, >, \neq, \leq, \geq \) applied to operands that are constants or attribute names or component numbers.
    • These are the **basic (atomic) clauses** of the conditions.
  – Boolean combinations \( (\land, \lor, \neg) \) of basic clauses.
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:**

\[
E := R, S, \ldots | (E_1 \cup E_2) | (E_1 - E_2) | (E_1 \times E_2) | \pi_L(E) | \sigma_{\Theta}(E),
\]

where

- R, S, \ldots are relation schemas
- L is a list of attributes
- \Theta is a condition.
Strength from Unity and Combination

• By itself, each basic relational algebra operation has limited expressive power, as it carries out a specific and rather simple task.

• When used in combination, however, the five relational algebra operations can express interesting and, quite often, rather complex queries.

• Derived relational algebra operations are operations on relations that are expressible via a relational algebra expression (built from the five basic operators).
Natural Join

**Definition:** Let $A_1, \ldots, A_k$ be the common attributes of two relation schemas $R$ and $S$. Then

$$R \bowtie S = \pi_{<\text{list}>} (\sigma_{R.A_1=S.A_1 \land \ldots \land R.A_1=S.A_k} (R \times S)),$$

where $<\text{list}>$ contains all attributes of $R \times S$, except for $S.A_1, \ldots, S.A_k$ (in other words, duplicate columns are eliminated).

**Example:** Given

$TEACHES(\text{fac-name}, \text{course}, \text{term})$ and $ENROLLS(\text{stud-name}, \text{course}, \text{term})$,

we want to obtain

$TAUGHT-BY(\text{stud-name}, \text{course}, \text{term}, \text{fac-name})$

Then

$TAUGHT-BY = ENROLLS \bowtie TEACHES$
Independence of the Basic Operations

• **Question:** Are all five basic relational algebra operations really needed? Can one of them be expressed in terms of the other four?

• **Theorem:** Each of the five basic relational algebra operations is independent of the other four, that is, it cannot be expressed by a relational algebra expression that involves only the other four.

**Proof Idea:** For each relational algebra operation, we need to discover a property that is possessed by that operation, but is not possessed by any relational algebra expression that involves only the other four operations.
SQL vs. Relational Algebra

<table>
<thead>
<tr>
<th>SQL</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Projection ( \pi )</td>
</tr>
<tr>
<td>FROM</td>
<td>Cartesian Product ( \times )</td>
</tr>
<tr>
<td>WHERE</td>
<td>Selection ( \sigma )</td>
</tr>
</tbody>
</table>

Semantics of SQL via interpretation to Relational Algebra

\[
\text{SELECT } R_{i_1}.A_1, \ldots, R_{i_m}.A.m \\
\text{FROM } R_1, \ldots, R_K \\
\text{WHERE } \psi = \pi_{R_{i_1}.A_1, \ldots, R_{i_m}.A.m} (\sigma_\psi (R_1 \times \ldots \times R_K))
\]
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 (FO Logic for Databases)

- **First-order variables**: $x, y, z, \ldots, x_1, \ldots, x_k, \ldots$
  - They range over values that may occur in tables.
- **Relation symbols**: $R, S, T, \ldots$ 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)
  - $(x \text{ op } y)$, where op is one of $=, \neq, <, >, \leq, \geq$
  - $(x \text{ op } c)$, where $c$ is a constant and op is one of $=, \neq, <, >, \leq, \geq$.
- **Relational Calculus Formulas**:
  - Every atomic formula is a relational calculus formula.
  - If $\phi$ and $\psi$ are relational calculus formulas, then so are:
    - $(\phi \land \psi), (\phi \lor \psi), \neg \psi, (\phi \rightarrow \psi)$ (propositional connectives)
    - $(\exists x \phi)$ (existential quantification)
    - $(\forall x \phi)$ (universal quantification).
Relational Calculus as a Query Language

Definition:
• A relational calculus expression is an expression of the form
  \{ (x_1,...,x_k) : \varphi(x_1,...,x_k) \},
  where \varphi(x_1,...,x_k) is a relational calculus formula with x_1,...,x_k as its free variables.
• When applied to a relational database D, this relational calculus expression returns the k-ary relation that consists of all k-tuples (a_1,...,a_k) that make the formula “true” on D.

Example: The relational calculus expression
  \{ (x,y) : \exists z (E(x,z) \land E(z,y)) \}
returns the set P of all pairs of nodes (a,b) that are connected via a path of length 2.
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 \( \varphi \), 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 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
  $$\{(x_1, \ldots, x_k) : S(x_1, \ldots, x_k) \land \neg R(x_1, \ldots, x_k)\}$$
  is “domain independent”.
Active Domain and Active Domain Interpretation

Definition:

- The active domain \( \text{adom}(D) \) of a relational database instance \( D \) is the set of all values that occur in the relations of \( D \).

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

\[
\varphi_{\text{adom}}(D)
\]

is the result of evaluating \( \varphi(x_1,\ldots,x_k) \) over \( \text{adom}(D) \) and \( D \), i.e.,

- all variables and quantifiers are assumed to range over \( \text{adom}(D) \);
- the relation symbols in \( \varphi \) are interpreted by the relations in \( D \).
Equivalence of Relational Algebra and Calculus

**Theorem:** If \( q \) is a \( k \)-ary query, then the following statements are equivalent:

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

2. There is a relational calculus formula \( \psi \) such that \( q(D) = \psi_{\text{adom}}(D) \) (in other words, \( q \) is expressible in relational calculus under the active domain interpretation).
Equivalence of Relational Algebra and Calculus

Proof (Sketch):

1. $\Rightarrow$ 2. By a straightforward induction on the construction of relational algebra expressions.

Note: Projection $\pi$ is simulated using $\exists$

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 $D$, we have that $\text{adom}(D) = E(D)$.
   - 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.
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 \( \forall y R(x,y) \).

- \( \forall y R(x,y) \equiv \neg \exists y \neg R(x,y) \)
- \( \text{adom}(D) = \pi_1(R) \cup \pi_2(R) \)

<table>
<thead>
<tr>
<th>Rel.Calc. formula ( \varphi )</th>
<th>Relational Algebra Expression for ( \varphi^{\text{adom}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg R(x,y) )</td>
<td>( (\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R )</td>
</tr>
<tr>
<td>( \exists y \neg R(x,y) )</td>
<td>( \pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R) )</td>
</tr>
<tr>
<td>( \neg \exists y \neg R(x,y) )</td>
<td>( (\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)) )</td>
</tr>
</tbody>
</table>
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 $D$ is a database instance over $S$, then $q(D)$ is a $k$-ary relation on $\text{dom}(D)$ that is invariant under isomorphisms (i.e., if $h: D \to F$ is an isomorphism, then $q(F) = h(q(D))$).

- A Boolean query on $S$ is a function $q$ defined on database instances over $S$ such that if $D$ is a database instance over $S$, then $q(D) = 0$ or $q(D) = 1$, and $q(D)$ 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?
Fundamental Algorithmic Problems about Queries

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

- **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 D, we have that \( q(D) = q'(D) \)?)

- **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 D, we have that \( q(D) \subseteq q'(D) \)?)
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 $q \equiv q \wedge q'$.
Undecidability of Equivalence and Containment

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

Proof: Use Trakhtenbrot’s Theorem (1949):

The Finite Validity Problem is undecidable.
– Finite Validity Problem \( \preceq \) Query Equivalence Problem
  • If \( \psi^* \) is a fixed finitely valid relational calculus sentence, then for every relational calculus sentence \( \phi \), we have that
    \[
    \phi \text{ is finitely valid } \iff \phi \equiv \psi^*.
    \]

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

Proof: Query Equivalence \( \preceq \) Query Containment, since

\[
q \equiv q' \iff q \subseteq q' \text{ and } q' \subseteq q.
\]
Complexity of the Query Evaluation Problem

The Query Evaluation Problem for Relational Calculus:
Given a relational calculus formula $\phi$ and a database instance $D$, find $\phi^{\text{adom}}(D)$.

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

Proof: QBF – Quantified Boolean Formulas

Show that

\[ \text{QBF} \preceq_p \text{Query Evaluation for Relational Calculus} \]

Given QBF \( \forall x_1 \exists x_2 \ldots \forall x_k \psi \)

- Let V and P be two unary relation symbols
- Obtain \( \psi^* \) from \( \psi \) by replacing \( x_i \) by \( P(x_i) \), and \( \neg x_i \) by \( \neg P(x_i) \)
- Let D be the database instance with \( V = \{0,1\} \), \( P = \{1\} \).
- Then the following database statements are equivalent:

- \( \forall x_1 \exists x_2 \ldots \forall x_k \psi \) is true
- \( \forall x_1 (V(x_1) \rightarrow \exists x_2 (V(x_2) \land (\ldots \forall x_k(V(x_k) \rightarrow \psi^*))\ldots)) \) is true on D.
Complexity of the Query Evaluation Problem

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

  **Proof (Hint):** Let $\varphi$ 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 $\varphi^{\text{adom}(D)}$, by exhaustively cycling over all possible interpretations of the $x_i$’s.
    This runs in time $O(n^m)$, where $n = |D|$ (size of $D$).
  
  - 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 $\text{adom}(I)$ written in binary (so $O(\log n)$ space is used in each block).
    - Maintain also $m$ counters in binary to keep track of the number of elements examined.

<table>
<thead>
<tr>
<th>$\forall x_1$</th>
<th>$\exists x_2$</th>
<th>...</th>
<th>$\forall x_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$ in $\text{adom}(I)$ written in binary</td>
<td>$a_2$ in $\text{adom}(I)$ written in binary</td>
<td>...</td>
<td>$a_m$ in $\text{adom}(I)$ written in binary</td>
</tr>
</tbody>
</table>
Complexity of the Query Evaluation Problem

- **Corollary:** The Query Evaluation Problem for Relational Algebra is PSPACE-complete.

**Proof:**
The translation of relational calculus to relational algebra yields a polynomial-time reduction of the Query Evaluation Problem for Relational Calculus to the Query Evaluation Problem for Relational Algebra.
Summary

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

• The Query Equivalence Problem for Relational Calculus is undecidable.

• The Query Containment Problem for Relational Calculus is undecidable.
The Query Evaluation Problem Revisited

- Since the Query Evaluation Problem for Relational Calculus is PSPACE-hard, there are no polynomial-time algorithms for this problem, unless PSPACE = P (which is considered highly unlikely).
- Let’s take another look at the exponential-time algorithm for this problem:
  - Let $\varphi$ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let $D$ be a database instance.
  - **Exponential Time Algorithm**: We can find $\varphi^{\text{dom}}(D)$, by exhaustively cycling over all possible interpretations of the $x_i$’s. This runs in time $O(n^m)$, where $n = |D|$.
  - So, the running time is $O(|D|^{|\varphi|})$, where $|D|$ is the size of $D$ and $|\varphi|$ is the size of the relational calculus formula $\varphi$.
  - This tells that the source of exponentiality is the formula size.
The Query Evaluation Problem Revisited

• **Theorem:** Let \( \varphi \) be a fixed relational calculus formula. Then the following problem is solvable in polynomial time: given a database instance \( D \), find \( \varphi^{\text{dom}}(D) \). In fact, this problem is in LOGSPACE.

**Proof:**
Let \( \varphi \) be a fixed relational calculus formula \( \forall x_1 \exists x_2 \ldots \forall x_m \psi \)

– The previous algorithm has running time \( O(|D|^{\lvert \varphi \rvert}) \), which is a polynomial, since now \( \lvert \varphi \rvert \) is a constant.

– Moreover, the algorithm can now be implemented using logarithmic-space only, since we need only maintain a constant number of memory blocks, each of logarithmic size.

<table>
<thead>
<tr>
<th>( \forall x_1 )</th>
<th>( \exists x_2 )</th>
<th>...</th>
<th>( \forall x_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 ) in ( \text{dom(I)} ) written in binary</td>
<td>( a_2 ) in ( \text{dom(I)} ) written in binary</td>
<td>...</td>
<td>( a_m ) in ( \text{dom(I)} ) written in binary</td>
</tr>
</tbody>
</table>
Vardi’s Taxonomy of the Query Evaluation Problem

M.Y Vardi, “The Complexity of Relational Query Languages”, 1982

• **Definition:** Let L be a database query language.
  – The **combined complexity of L** is the decision problem:
    given an L-sentence and a database instance D, is \( \varphi \) true on D? (does D satisfy \( \varphi \)?) (in symbols, does D \( \models \varphi \)?)
  
  – The **data complexity of L** is the family of the following decision problems \( P_\varphi \), where \( \varphi \) is an L-sentence:
    given a database instance D, does D \( \models \varphi \)?
  
  – The **query complexity of L** is the family of the following decision problems \( P_D \), where D is a database instance:
    given an L-sentence \( \varphi \), does D \( \models \varphi \)?
Vardi’s Taxonomy of the Query Evaluation Problem

**Definition:** Let $L$ be a database query language and let $C$ be a computational complexity class.

- The **data complexity of $L$ is in $C$** if for each $L$-sentence $\varphi$, the decision problem $P_\varphi$ is in $C$.

- The **data complexity of $L$ is $C$-complete** if it is in $C$ and there is an $L$-sentence $\varphi$ such that the decision problem $P_\varphi$ is $C$-complete.

- The **query complexity of $L$ is in $C$** if for every database $D$, the decision problem $P_D$ is in $C$.

- The **query complexity of $L$ is $C$-complete** if it is in $C$ and there is a database $D$ such that the decision problem $P_D$ is $C$-complete.
Vardi’s Taxonomy of the Query Evaluation Problem

Vardi’s “empirical” discovery:

For most query languages L:
- The data complexity of L is of lower complexity than both the combined complexity of L and the query complexity of L.
- The query complexity of L can be as hard as the combined complexity of L.
The Query Evaluation Problem for Relational Calculus

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combined Complexity</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>Query Complexity</td>
<td>- Is in PSPACE</td>
</tr>
<tr>
<td></td>
<td>- It can be PSPACE-complete</td>
</tr>
<tr>
<td>Data Complexity</td>
<td>In LOGSPACE</td>
</tr>
</tbody>
</table>

Complexity Classes
- PSPACE
- NP
- P
- NLOGSPACE
- LOGSPACE
Summary

- Relational Algebra and Relational Calculus have “essentially” the same expressive power.

- The Query Equivalence Problem for Relational Calculus is undecidable.

- The Query Containment Problem for Relational Calculus is undecidable.

- The Query Evaluation Problem for Relational Calculus is \( \text{PSPACE}\)-complete (combined / query complexity).
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 $\land$ 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 $\Theta$ 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 $\land$ 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**:
  $Q(x_1,\ldots,x_k) :-- R_1(u_1), \ldots, R_n(u_n)$, where
  - Each variable $x_i$ occurs in the right-hand side of the rule.
  - Each $u_i$ 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.
Examples of Conjunctive Queries

– **Path of Length 2:** (Binary query)
  \[(x,y) : \exists z (E(x,z) \land E(z,y))\]

  • 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)
  \[\exists x \exists y \exists z (E(x,y) \land E(y,z) \land E(z,x))\]

  • As a rule (the head has no variables)
    – \[Q :- E(x,z), E(z,y), E(z,x)\]
Conjunctive Queries

• Every natural join is a conjunctive query with no existentially quantified variables
  
  P(A,B,C), R(B,C,D) two relation symbols
  
  \[ P \bowtie R = \{(x,y,z,w) : P(x,y,z) \land R(y,z,w)\} \]

\[ q(x,y,z,w) :-- P(x,y,z), R(y,z,w) \]

(no variables are existentially quantified)

\[ \text{SELECT} \ P.A, P.B, P.C, R.D \]
\[ \text{FROM} \ P, R \]
\[ \text{WHERE} \ P.B = R.B \ \text{AND} \ P.C = R.C \]

• 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 $D$, find $q(D)$.
  
  - **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 $D$, we have that $q_1(D) \subseteq q_2(D)$)
    - Given two Boolean conjunctive queries $q_1$ and $q_2$,
      is it true that $q_1 \models q_2$? (that is, for all $D$, if $D \models q_1$, then $D \models q_2$)?

CQC is logical implication.
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
Homomorphisms

• Definition: Let D and F be two database instances over the same relational schema S. A homomorphism $h: D \rightarrow F$ is a function $h: \text{adom}(D) \rightarrow \text{adom}(F)$ such that for every relational symbol $P$ of S and every $(a_1,\ldots,a_m)$, we have that

$$\text{if } (a_1,\ldots,a_m) \in P^D \text{, then } (h(a_1), .., h(a_m)) \in P^F.$$ 

Note: The concept of homomorphism is a relaxation of the concept of isomorphism, since every isomorphism is also a homomorphism, but not vice versa.

• Example:

A graph $G = (V,E)$ is 3-colorable if and only if there is a homomorphism $h: G \rightarrow K_3$.
The Homomorphism Problem

- **Definition:** The Homomorphism Problem
  Given two database instances D and F, is there a homomorphism h: D → F?

- **Notation:** D → F denotes that a homomorphism from D to F exists.

- **Theorem:** The Homomorphism Problem is NP-complete.
  
  **Proof:** Easy reduction from 3-Colorability

  G is 3-colorable if and only if G → K₃.

- **Exercise:**
  Formulate 3SAT as a special case of the Homomorphism Problem.
The Homomorphism Problem

- **Note:** The Homomorphism Problem is a fundamental algorithmic problem:
  - **Satisfiability** can be viewed as a special case of it.
  - **k-Colorability** can be viewed as a special case of it.
  - Many AI problems, such as **planning**, can be viewed as a special case of it.
  - In fact, every **constraint satisfaction problem** can be viewed as a special case of the Homomorphism Problem (Feder and Vardi – 1993).
Homomorphism Problem & Conjunctive Queries

- **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**:
  - Both CQE and CQC are “equivalent” to the Homomorphism Problem.
  - The link is established by bringing into the picture
    - Canonical conjunctive queries and
    - Canonical database instances.