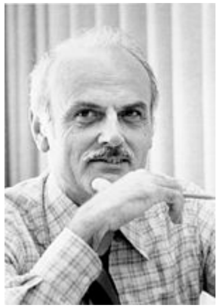
Relational Model

Relational Model

- Many ad hoc models before 1970
  - Hard to work with
  - Hard to reason about



- **1970: Relational Model by Edgar Frank Codd**
  - Data are stored in relations (or tables)
  - Queried using a declarative language
  - DBMS converts declarative queries into procedural queries that are optimized and executed

- Key Advantages
  - Simple and clean mathematical model (based on logic)
  - Separation of declarative and procedural

Relational Databases

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants

VIE, LHR, …

BA, U2, …

Vienna, London, …

Relational Databases

Relations

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants

VIE, LHR, …

BA, U2, …

Vienna, London, …

Relational Databases

Relations

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Tuples

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants

VIE, LHR, …

BA, U2, …

Vienna, London, …

Relational atoms

Flight(LHR,EDI,BA)

Airport(LGW,London)

Querying: Relational Algebra

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{BA, U2, OS}

π_{airline} Flight

Querying: Relational Algebra

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$\Downarrow$

{LHR, LGW}

$\pi_{code}\ (\sigma_{city=\text{'London'}}\ \text{Airport})$

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$$\pi_{\text{airline}}\ ((\text{Flight} \bowtie_{\text{origin=code}} (\sigma_{\text{city='London'}}\ \text{Airport})) \bowtie_{\text{destination=code}} (\sigma_{\text{city='Glasgow'}}\ \text{Airport}))$$

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Aux	origin	destination	airline	code	city	code	city
	LGW	GLA	U2	LGW	London	GLA	Glasgow

⇩

{U2}

$$\pi_{airline}\ \underbrace{((Flight \bowtie_{origin=code} (\sigma_{city=\text{'London'}}\ Airport)) \bowtie_{destination=code} (\sigma_{city=\text{'Glasgow'}}\ Airport))}_{\text{defines the auxiliary relation Aux}}$$

Relational Algebra

- **Selection: σ**
- **Projection: π**
- **Cross product:** $\times$
- Natural join: $\bowtie$
- **Rename: ρ**
- **Difference:** \
- **Union:** ∪
- Intersection: ∩

in bold are the primitive operators

Formal definitions can be found in any database textbook

Querying: Domain Relational Calculus

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{BA, U2, OS}

$\{z \mid \exists x \exists y \; Flight(x,y,z)\}$

Querying: Domain Relational Calculus

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

$\{x \mid \exists y\ \mathrm{Airport}(x,y) \ \wedge\ y = \mathrm{London}\}$

Querying: Domain Relational Calculus

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

{U2}

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$$\{z \mid \exists x \exists y \exists u \exists v\ \mathrm{Airport}(x,u) \ \wedge\ u = \mathrm{London} \ \wedge\ \mathrm{Airport}(x,u) \ \wedge\ u = \mathrm{London}\} \ \wedge\ \mathrm{Flight}(x,y,z)\}$$

Domain Relational Calculus

$$\{x_1,\ldots,x_k \mid \phi\}$$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\ R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

Domain Relational Calculus

$$\{x_1,\ldots,x_k \mid \phi\}$$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\ R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

domain = $\{1,2,3\}$

D = {R(1,1), R(1,2)}

Ans = { }

Domain Relational Calculus

$$\{x_1,\ldots,x_k \mid \phi\}$$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

domain = $\{1,2\}$

D = $\{R(1,1), R(1,2)\}$

Ans = $\{1\}$

Domain Relational Calculus

$$\{x_1,\ldots,x_k \mid \phi\}$$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

...thus, we adopt the active domain semantics - quantified variables range over

the active domain, i.e., the constants occurring in the input database

Algebra = Calculus

A fundamental theorem (assuming the active domain semantics):

Theorem: The following query langauges are equally expressive

- Relational Algebra (**RA**)
- Domain Relational Calculus (**DRC**)
- Tuple Relational Calculus (**TRC**)

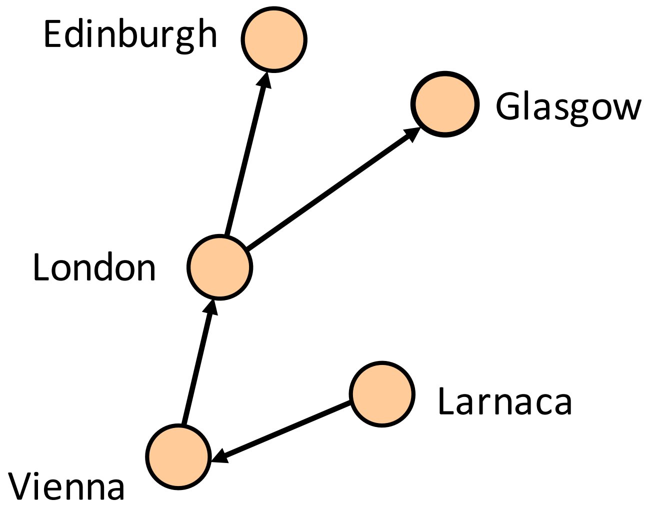
Note: Tuple relational calculus is the declarative language introduce by Codd. Domain relational calculus has been introduced later as a formalism closer to first-order logic

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



{ | ∃x∃y∃z∃w∃v Airport(x,Vienna) ∧ Airport(y,Glasgow) ∧

Flight(x,z,w) ∧ Flight(z,y,v)

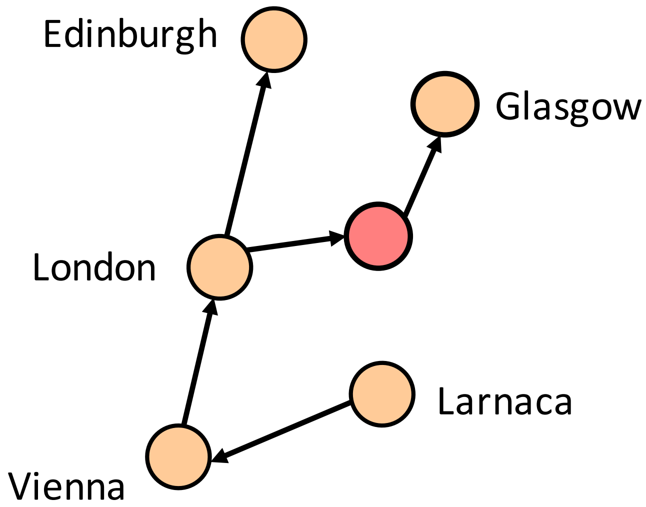
YES

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$$\{ \mid \exists x \exists y \exists z \exists w \exists v \ \ \text{Airport}(x,\text{Vienna}) \ \wedge \ \text{Airport}(y,\text{Glasgow}) \ \wedge \ \text{Flight}(x,z,w) \ \wedge \ \text{Flight}(z,y,v)$$

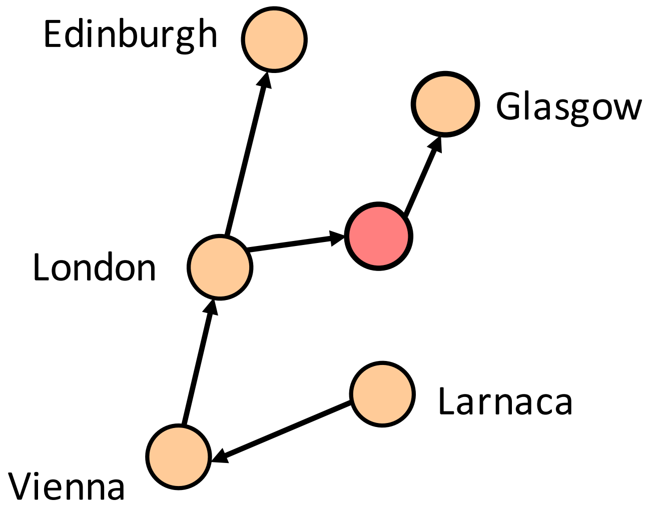
NO

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$\{ \mid \exists x \exists y \exists z \exists w \exists v\ \text{Airport}(x,\text{Vienna}) \land \text{Airport}(y,\text{Glasgow}) \land$

$\exists z_1 \exists w_1 \quad \text{Flight}(x,z,w) \quad \text{Flight}(z,y,v)$

$\land\ \text{Flight}(z,z_1,w_1)\ \land$

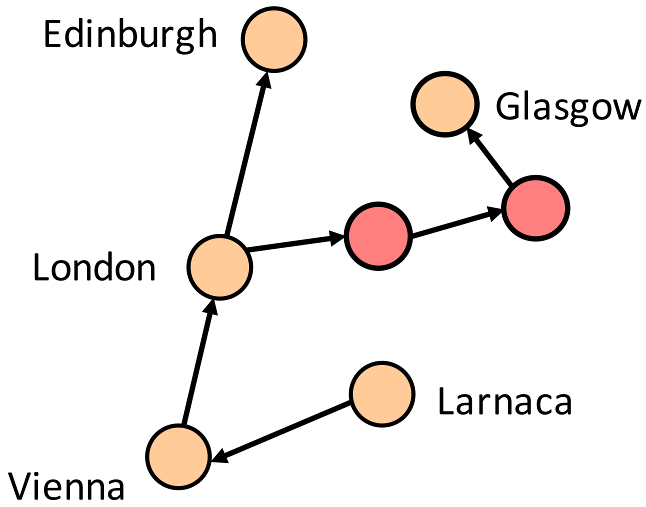
YES

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$\{\ \mid\ \exists x \exists y \exists z \exists w \exists v\ \text{Airport}(x,\text{Vienna}) \ \wedge\ \text{Airport}(y,\text{Glasgow}) \ \wedge$

$\exists z_1 \exists w_1 \quad \text{Flight}(x,z,w) \quad \text{Flight}(z,y,v)$

$\wedge\ \text{Flight}(z,z_1,w_1)\ \wedge$

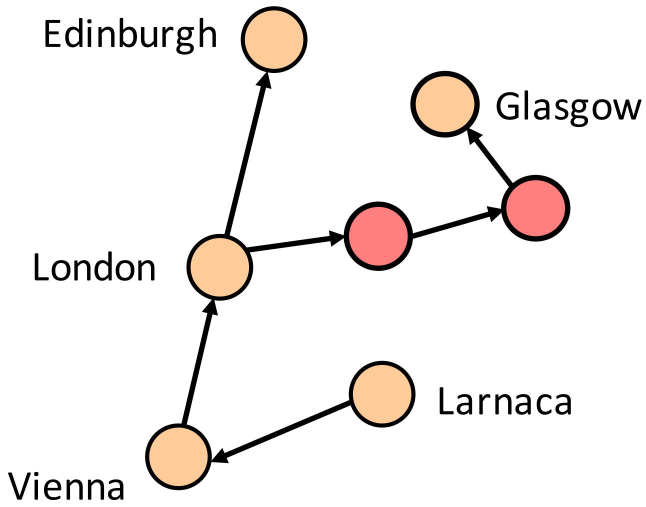
NO

Quiz!

Is Glasgow reachable from Vienna?

Flight	**origin**	**destination**	**airline**
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	**code**	**city**
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Recursive query - not expressible in **RA/DRC/TRC**

(unless we bound the number of intermediate stops)

Complexity of Query Languages

- The goal is to understand the complexity of evaluating a query over a database
- Our main technical tool is complexity theory
- What to measure? Queries may have a large output, and it would be unfair to count the output as “complexity”
- We therefore consider the following decision problems:
  - Query Output Tuple (QOT)
  - Boolean Query Evaluation (BQE)

A Few Words on Complexity Theory

details can be found in the standard textbooks

see also notes on the webpage of the course

Complexity Classes

Consider a function $f : N \to N$

$$\mathrm{TIME}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some DTM in time } O(f(n))\}$$

$$\mathrm{NTIME}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some NTM in time } O(f(n))\}$$

$$\mathrm{SPACE}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some DTM using space } O(f(n))\}$$

$$\mathrm{NSPACE}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some NTM using space } O(f(n))\}$$

Complexity Classes

- We can now recall the standard time and space complexity classes:

PTIME	=	$\bigcup_{k>0}$ TIME(n^k)
NP	=	$\bigcup_{k>0}$ NTIME(n^k)
EXPTIME	=	$\bigcup_{k>0}$ TIME(2^{n^k})
NEXPTIME	=	$\bigcup_{k>0}$ NTIME(2^{n^k})
LOGSPACE	=	SPACE(log n)
NLOGSPACE	=	NSPACE(log n)
PSPACE	=	$\bigcup_{k>0}$ SPACE(n^k)
EXPSPACE	=	$\bigcup_{k>0}$ SPACE(2^{n^k})

these definitions are relying on two-tape Turing machines with a read-only and a read/write tape

- For every complexity class C we can define its complementary class coC

Relationship Among Complexity Classes

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NP}, \text{coNP} \subseteq$$
$$\text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME}, \text{coNEXPTIME} \subseteq \cdots$$

Some useful notes:

- For a deterministic complexity class C, coC = C
- coNLOGSPACE = NLOGSPACE
- It is generally believed that PTIME $\neq$ NP, but we don't know
- PTIME $\subset$ EXPTIME $\Rightarrow$ at least one containment between them is strict
- PSPACE = NPSPACE, EXPSPACE = NEXPSPACE, etc.
- But, we don't know whether LOGSPACE = NLOGSPACE

Complete Problems

- These are the hardest problems in a complexity class
- A problem that is complete for a class C, it is unlikely to belong in a lower class

- A problem Π is complete for a complexity class C, or simply C-complete, if:
    1. $\Pi \in C$
    2. Π is C-hard, i.e., every problem $\Pi' \in C$ can be efficiently reduced to Π

there exists a logspace algorithm that computes a function f such that

$\mathbf{w} \in \Pi'$ iff $f(\mathbf{w}) \in \Pi$ - in this case we write $\Pi' \leq_L \Pi$

- To show that Π is C-hard it suffices to reduce some C-hard problem Π' to it

Some Complete Problems

- NP-complete
  - SAT (satisfiability of propositional formulas)
  - Many graph-theoretic problems (e.g., 3-colorability)
  - Traveling salesman
  - etc.

- PSPACE-complete
  - Quantified SAT (or simply QSAT)
  - Equivalence of two regular expressions
  - Many games (e.g., Geography)
  - etc.

Back to Query Languages

Complexity of Query Languages

- The goal is to understand the complexity of evaluating a query over a database
- Our main technical tool is complexity theory
- What to measure? Queries may have a large output, and it would be unfair to count the output as "complexity"
- We therefore consider the following decision problems:
  - Query Output Tuple (QOT)
  - Boolean Query Evaluation (BQE)

Complexity of Query Languages

Some useful notation:

- Given a database D, and a query Q, Q(D) is the answer to Q over D
- **adom**(D) is the active domain of D - the constants occurring in D
- We write Q/k for the fact that the arity of Q is $k \geq 0$

L is some query language; for example, **RA**, **DRC**, etc. - we will see several query languages

QOT(**L**)

Input: a database D, a query Q/k ∈ **L**, a tuple of constants **t** ∈ **adom**(D)^k

Question: **t** ∈ Q(D)?

Complexity of Query Languages

Some useful notation:

- Given a database D, and a query Q, $Q(D)$ is the answer to Q over D
- **adom**(D) is the active domain of D - the constants occurring in D
- We write Q/k for the fact that the arity of Q is $k \geq 0$

L is some query language; for example, **RA**, **DRC**, etc. - we will see several query languages

BQE(**L**)

Input: a database D, a Boolean query $Q \in$ **L**

Question: is $Q(D)$ non-empty?

Complexity of Query Languages

QOT(**L**)

Input: a database D, a query Q/k ∈ **L**, a tuple of constants **t** ∈ **adom**(D)^k

Question: **t** ∈ Q(D)?

BQE(**L**)

Input: a database D, a Boolean query Q ∈ **L**

Question: is Q(D) non-empty?

Theorem: QOT(**L**) $\equiv_L$ BQE(**L**), where **L** ∈ {**RA**, **DRC**, **TRC**}

($\equiv_L$ means logspace-equivalent)

Complexity of Query Languages

(let us show this for domain relational calculus)

Theorem: QOT(**DRC**) $\equiv_L$ BQE(**DRC**)

Proof: ($\leq_L$) Consider a database D, a k-ary query $Q = \{x_1,\ldots,x_k \mid \phi\}$, and a tuple $(t_1,\ldots,t_k)$

Let $Q_{bool} = \{ \mid \exists x_1 \cdots \exists x_k (\phi \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge \cdots \wedge x_k = t_k) \}$

Clearly, $(t_1,\ldots,t_k) \in Q(D)$ iff $Q_{bool}(D)$ is non-empty

($\geq_L$) Trivial - a Boolean domain RC query is a domain RC query

Complexity Measures

- Combined complexity - both D and Q are part of the input
- Query complexity - fixed D, input Q
- Data complexity - input D, fixed Q

BQE[D](**L**)

Input: a Boolean query Q ∈ **L**

Question: is Q(D) non-empty?

BQE[Q](**L**)

Input: a database D

Question: is Q(D) non-empty?

Complexity of **RA**, **DRC**, **TRC**

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**} the following hold:

- BQE(**L**) is PSPACE-complete (combined complexity)
- BQE[D](**L**) is PSPACE-complete, for a fixed database D (query complexity)
- BQE[Q](**L**) is in LOGSPACE, for a fixed query Q ∈ **L** (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in Q and logarithmic space in D
- Reduction from QSAT (a standard PSPACE-hard problem)

Evaluating (Boolean) DRC Queries

Eval(D,ϕ) - for brevity we write ϕ instead of $\{ \mid \phi\}$

- **If** $\phi = R(t_1,\ldots,t_k)$, **then** YES iff $R(t_1,\ldots,t_k) \in D$
- **If** $\phi = \psi_1 \wedge \psi_2$, **then** YES iff Eval(D,ψ_1) = YES and Eval(D,ψ_2) = YES
- **If** $\phi = \neg\psi$, **then** NO iff Eval(D,ψ) = YES
- **If** $\phi = \exists x\ \psi(x)$, **then** YES iff for some $t \in$ **adom**(D), Eval(D,$\psi(t)$) = YES

Lemma: It holds that

- Eval(D,ϕ) always terminates - this is trivial
- Eval(D,ϕ) = YES iff Q(D) is non-empty, where $Q = \{ \mid \phi\}$
- Eval(D,ϕ) uses $O(|\phi| \cdot \log |\phi| + |\phi|^2 \cdot \log |D|)$ space

Complexity of **RA**, **DRC**, **TRC**

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**} the following hold:

- BQE(**L**) is PSPACE-complete (combined complexity)
- BQE[D](**L**) is PSPACE-complete, for a fixed database D (query complexity)
- BQE[Q](**L**) is in LOGSPACE, for a fixed query Q ∈ **L** (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in Q and logarithmic space in D
- Reduction from QSAT (a standard PSPACE-hard problem)

Other Important Algorithmic Problems

SAT(**L**)

Input: a query $Q \in \mathbf{L}$

Question: is there a (finite) database D such that $Q(D)$ is non-empty?

EQUIV(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every (finite) database D?

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?

Other Important Algorithmic Problems

SAT(**L**)

Input: a query Q ∈ **L**

Question: is there a (finite) database D such that Q(D) is non-empty?

EQUIV(**L**)

Input: two

Question: …abase D?

these problems are important

for optimization purposes

CONT(**L**)

Input: two queries Q_1 ∈ **L** and Q_2 ∈ **L**

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?

Other Important Algorithmic Problems

SAT(**L**)

Input: a query Q ∈ **L**

Question: is there a (finite) database D such that Q(D) is non-empty?

- If the answer is no, then the input query Q makes no sense
- Query evaluation becomes trivial - the answer is always NO!

Other Important Algorithmic Problems

EQUIV(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every (finite) database D?

- Replace a query Q_1 with a query Q_2 that is easier to evaluate
- But, we have to be sure that $Q_1(D) = Q_2(D)$ for every database D

Other Important Algorithmic Problems

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?

- Approximate a query Q_1 with a query Q_2 that is easier to evaluate
- But, we have to be sure that $Q_1(D) \subseteq Q_2(D)$ for every database D

SAT is Undecidable

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**}, SAT(**L**) is undecidable

Proof hint: By reduction from the halting problem.

Given a Turing machine M, we can construct a query Q_M ∈ **L** such that:

M halts on the empty string iff there exists a database D such that Q(D) is non-empty

Note: Actually, this result goes back to the 1950 when Boris A. Trakhtenbrot proved that the problem of deciding whether a first-order sentence has a finite model is undecidable

EQUIV and CONT are Undecidable

An easy consequence of the fact that SAT is undecidable is that:

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**}, EQUIV(**L**) and CONT(**L**) are undecidable

Proof: By reduction from the complement of SAT(**L**)

- Consider a query Q ∈ **L** - i.e., an instance of SAT(**L**)
- Let Q' be a query that is unsatisfiable, i.e., Q'(D) is empty for every D
- For example, when **L** = **DRC**, Q' can be the query $\{ \mid \exists x\, R(x) \land \neg R(x)\}$
- Clearly, Q is unsatisfiable iff $Q \equiv Q'$ (or even $Q \subseteq Q'$)

Recap

- The main languages for querying relational databases are:
  - Relational Algebra (**RA**)
  - Domain Relational Calcuclus (**DRC**)
  - Tuple Relational Calculus (**TRC**)

RA = DRC = TRC

(under the active domain semantics)

- Evaluation is decidable, and highly tractable in data complexity
  - **Foundations of the database industry**
  - The core of SQL is equally expressive to **RA**/**DRC**/**TRC**

- Satisfiability, equivalence and containment are undecidable
  - **Perfect query optimization is impossible**