

Databases

Relational Calculus

Pierre Senellart



11 February 2026

Plan

Relational Calculus

Definitions

Codd's theorem

Complexity of Query Evaluation

Static Analysis of Queries

Conclusion

Relational calculus

- Logical language to express queries
- First-order logic formula, without function symbols, and with relation symbols the labels of the database schema (plus comparison predicates)
- Unnamed, untyped perspective
- Fix:
 - A set $\mathcal{X}$ of variables
 - A set $\mathcal{V}$ of values
 - A database schema S

Relational calculus: Syntax

- For every relation $R \in S$ of arity n , for every $(\alpha_1, \dots, \alpha_n) \in (\mathcal{X} \cup \mathcal{V})^n$: $R(\alpha_1, \dots, \alpha_n) \in \text{FO}$
- Also allow equality predicate, i.e., $\alpha = \alpha' \in \text{FO}$ for $(\alpha, \alpha') \in (\mathcal{X} \cup \mathcal{V})^2$
- For every $(\varphi_1, \varphi_2) \in \text{FO}^2$, for every $x \in \mathcal{X}$:
 - $\varphi_1 \wedge \varphi_2 \in \text{FO}$
 - $\varphi_1 \vee \varphi_2 \in \text{FO}$
 - $\neg \varphi_1 \in \text{FO}$
 - $\forall x \varphi_1 \in \text{FO}$
 - $\exists x \varphi_1 \in \text{FO}$
- **Free variables** of $\varphi \in \text{FO}$: variables x appearing in φ and not qualified by a $\forall x$ or a $\exists x$
- One writes a relational calculus **query** in the form $Q(x_1, \dots, x_m) = \varphi$ where $x_1, \dots, x_m$ are free variables of φ

Relational calculus: Semantics

- A relational calculus query on schema S can be seen as a **function** with input a database D over S and producing a relation as output
- $\text{adom}(D)$: **active domain** of D , set of values in D
- If $Q(x_1, \dots, x_n) = \varphi$ is a calculus query over S and D a database over S , then:

$$Q(D) = \{ (v_1, \dots, v_n) \in (\text{adom}(D))^n \mid D \models \varphi[x_1/v_1, \dots, x_n/v_n] \}$$

where $D \models \varphi$ is defined inductively:

- $D \models R(u_1, \dots, u_m) \iff R(u_1, \dots, u_m) \in D$
- $D \models \varphi_1 \wedge \varphi_2 \iff (D \models \varphi_1) \wedge (D \models \varphi_2)$
- $D \models \varphi_1 \vee \varphi_2 \iff (D \models \varphi_1) \vee (D \models \varphi_2)$
- $D \models \neg \varphi_1 \iff D \not\models \varphi_1$
- $D \models \forall x \varphi_1 \iff \forall v \in \text{adom}(D) D \models \varphi_1[x/v]$
- $D \models \exists x \varphi_1 \iff \exists v \in \text{adom}(D) D \models \varphi_1[x/v]$

Relational calculus: Example

Guest			Reservation				
id	name	email	id	guest	room	arrival	nights
1	John Smith	john.smith@gmail.com	1	1	504	2017-01-01	5
2	Alice Black	alice@black.name	2	2	107	2017-01-10	3
3	John Smith	john.smith@ens.fr	3	3	302	2017-01-15	6
			4	2	504	2017-01-15	2
			5	2	107	2017-01-30	1

$$\begin{aligned} & \Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation})) \\ & \cup \Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation})) \end{aligned}$$

$$\begin{aligned} Q(r) = & \exists i_1 \exists a_1 \exists n_1 \exists i_2 \exists g_2 \exists n_2 \\ & \text{Reservation}(i_1, 2, r, a_1, n_1) \vee \\ & \text{Reservation}(i_2, g_2, r, 2017-01-15, n_2) \end{aligned}$$

Subclasses of queries

- **Conjunctive query (CQ)**: relational calculus query without $\vee, \neg, \forall$
- **Positive query (PQ)**: relational calculus query without $\neg, \forall$
- **Union of conjunctive queries (UCQ)**: special case of positive query where the $\vee$ and $\wedge$ form a DNF formula
- **Boolean query**: a query with **no free variable**

Plan

Relational Calculus

Definitions

Codd's theorem

Complexity of Query Evaluation

Static Analysis of Queries

Conclusion

Codd's theorem

Theorem ([Codd, 1972])

The relational algebra and the relational calculus are equivalent:

- *for every relational algebra query q over a schema S , there exists a relational calculus query Q over S such that for every database D over S , $q(D) = Q(D)$*
- *for every relational calculus query Q over a schema S , there exists a relational algebra query q over S such that for every database D over S , $q(D) = Q(D)$*

Furthermore, translating from one formalism to the other can be done in polynomial time.

How to prove Codd's theorem

Algebra $\rightarrow$ Calculus Show how each algebra operator can be
rewritten in the calculus

Calculus $\rightarrow$ Algebra Write an algebra query that produces all
value of the active domain; use that algebra query
to inductively **rewrite** each calculus expression to
an algebra expression

Why is this important?

- Allows using a **declarative formalism** to specify queries: logics... or SQL
- These queries are then compiled via Codd's transformation into an **algebraic formalism**
- Algebraic queries are then **optimized**, by using the properties of the relational algebra (transformation rules, e.g., pushing selection within joins, exploiting associativity of joins, etc.)
- Optimized queries can then be **evaluated**, by exploiting the fact that each operator of the relational algebra can easily be implemented (in several different ways, to be chosen based on a cost function)
- This is RDBMS Implementation 101, a main reason of the success of RDBMSs!

What about subclasses of FO?

- CQs are **equivalent** to the relational algebra without $\cup$ and $\setminus$, and where σ does not feature disjunction
- UCQs are **equivalent** to PQs (but exponential blow-up), and equivalent to the relational algebra without $\setminus$

Plan

Relational Calculus

Complexity of Query Evaluation

Definitions

Relational Calculus

Static Analysis of Queries

Conclusion

Query evaluation

- Query Q in some query language (e.g., FO) – we will use a logical formalism here
- Database D (always **finite**!)
- **Query evaluation**: Computing $Q(D)$
- **Complexity** of this problem?
- To simplify the study of complexity, we often assume that Q is a **Boolean** query, i.e., it returns $\top$ or $\perp$

Data complexity

For some fixed Q , what is the complexity of computing $Q(D)$ in terms of the **size of the database D** ?

Combined complexity

For some query language $\mathcal{Q}$, what is the complexity of computing $Q(D)$ in terms of the size of the query $Q \in \mathcal{Q}$ and of the database D ?

Complexity classes

- We restrict to **Boolean** problems (returning $\top$ or $\perp$)
- Set of all problems solvable by a **resource-constrained computing method**:
- For example:
 - PTIME**: deterministic Turing machine in polynomial time
 - NP**: non-deterministic Turing machine in polynomial time
 - PSPACE**: deterministic Turing machine in polynomial space
 - AC⁰**: Boolean circuit of polynomial size and constant depth
- We know that: $AC^0 \subsetneq PTIME \subseteq NP \subseteq PSPACE$
- Open whether $PSPACE \subseteq PTIME$ (!)

Membership and hardness for a class

- A problem P **belongs** to a complexity class $\mathcal{C}$ (or **in** $\mathcal{C}$) if it is **solvable** by the corresponding resource-constrained computing method
- A problem P is **hard** for a complexity class $\mathcal{C}$ (or **$\mathcal{C}$ -hard**) if there exists a **reduction** that transforms whatever problem $P' \in \mathcal{C}$ into an instance of the problem P
- **complete**: in $\mathcal{C} + \mathcal{C}$ -hard
- Several ways to define reductions
- Here, we assume that there exists a function computable **in polynomial time** that **transforms** one instance I' of problem P' into an instance I of P such that $P(I) = P'(I')$

Descriptive complexity

- A query language $\mathcal{Q}$ **captures** a complexity class $\mathcal{C}$ if:
 - For all $Q \in \mathcal{Q}$, query evaluation of query Q is in $\mathcal{C}$ (data complexity)
 - For all problem P in $\mathcal{C}$, there exists a query $Q \in \mathcal{Q}$ such that evaluating Q **exactly solves** P (without a reduction)!
- If $\mathcal{Q}$ captures $\mathcal{C}$ and if $\mathcal{C}$ has problems that are complete for $\mathcal{C}$, then there exists $Q \in \mathcal{Q}$ such that Q is $\mathcal{C}$ -complete, but **the converse is not true**

Plan

Relational Calculus

Complexity of Query Evaluation

Definitions

Relational Calculus

Static Analysis of Queries

Conclusion

Data complexity

Theorem

*FO evaluation is **PTIME** in data complexity.*

Proof.

By rewriting in prenex normal form and naive evaluation.



FO does not capture the whole of PTIME

Theorem

One cannot compute in FO that a relation containing a total order has an even number of elements, or that a graph is connected.

Fairly complex to prove, relies on Ehrenfeucht–Fraïssé games (see [Libkin, 2004]).

Why FO cannot express connectivity

- First-order formulas have **fixed quantifier depth**.
- No mechanism for **recursion** or unbounded iteration.

Why FO cannot express connectivity

- First-order formulas have **fixed quantifier depth**.
- No mechanism for **recursion** or unbounded iteration.

Intuition

A first-order formula can only inspect neighborhoods of bounded radius in a structure.

Why FO cannot express connectivity

- First-order formulas have **fixed quantifier depth**.
- No mechanism for **recursion** or unbounded iteration.

Intuition

A first-order formula can only inspect neighborhoods of bounded radius in a structure.

Consequence

Properties such as graph connectivity require propagating information arbitrarily far: this cannot be expressed in FO.

Data complexity, more precise

Theorem

FO evaluation is AC^0 in data complexity.

Proof.

By rewriting to the relational algebra.



Data complexity, more precise

Theorem

FO evaluation is AC^0 in data complexity.

Proof.

By rewriting to the relational algebra.



Implication

- No deep sequential computation required
- Explains high parallelizability of SQL engines

Data complexity, more precise

Theorem

FO evaluation is AC^0 in data complexity.

Proof.

By rewriting to the relational algebra.



Implication

- No deep sequential computation required
- Explains high parallelizability of SQL engines

Declarative does not mean expensive.

Combined complexity

Theorem

*FO evaluation is **PSPACE-complete** in combined complexity.*

Proof.

Membership in PSPACE easy. Hardness for PSPACE from the QSAT problem: is a quantified Boolean formula satisfiable? $\square$

Plan

Relational Calculus

Complexity of Query Evaluation

Static Analysis of Queries

Containment and Equivalence

Relational Calculus

Conclusion

Query optimization

- **Goal:** Given a query q in some query language Q and a database D , find a query **equivalent to q on D** and faster to evaluate on D
- **Here:** Q in the relational calculus (or a fragment thereof), and one looks for a query faster **on whatever database** (we do not look at D , we perform **static analysis**)
- **In actual RDBMSs:** Q is the set of **query execution plans** (a specialization of the relational algebra where implementations are chosen for each operator) and statistics on D are used

Global optimization

- We consider **global** optimization techniques, considering a query in its entirety (techniques on execution plans are more local, e.g., local rewritings)
- We formally define:

Equivalence: $q \equiv q'$ if for all database D , $q(D) = q'(D)$

Minimality: q' is the “best” query equivalent to q in $\mathcal{Q}$

Containment and equivalence

Definition

A query q is **contained** in a query q' (denoted $q \sqsubseteq q'$) if for all database D , $q(D) \subseteq q'(D)$

Containment and equivalence

Definition

A query q is **contained** in a query q' (denoted $q \sqsubseteq q'$) if for all database D , $q(D) \subseteq q'(D)$

Proposition

$q \equiv q'$ iff $q \sqsubseteq q'$ and $q' \sqsubseteq q$.

Proof.

Immediate. □

Example of containment

Consider the schema with relations $R(x, y)$ and $S(y)$.

Queries:

$$q(x) = \exists y R(x, y)$$

$$q'(x) = \exists y (R(x, y) \wedge S(y))$$

$$q''(x) = \exists y \exists z (R(x, y) \wedge S(y) \wedge S(z))$$

Example of containment

Consider the schema with relations $R(x, y)$ and $S(y)$.

Queries:

$$q(x) = \exists y R(x, y)$$

$$q'(x) = \exists y (R(x, y) \wedge S(y))$$

$$q''(x) = \exists y \exists z (R(x, y) \wedge S(y) \wedge S(z))$$

Proposition

- $q' \sqsubseteq q$ but $q \not\sqsubseteq q'$.

Example of containment

Consider the schema with relations $R(x, y)$ and $S(y)$.

Queries:

$$q(x) = \exists y R(x, y)$$

$$q'(x) = \exists y (R(x, y) \wedge S(y))$$

$$q''(x) = \exists y \exists z (R(x, y) \wedge S(y) \wedge S(z))$$

Proposition

- $q' \sqsubseteq q$ but $q \not\sqsubseteq q'$.
- $q' \equiv q''$.

Plan

Relational Calculus

Complexity of Query Evaluation

Static Analysis of Queries

Containment and Equivalence

Relational Calculus

Conclusion

Satisfiability in the relational calculus

Definition

A Boolean relational calculus query q is **satisfiable** if there exists a (finite) database D such that $D \models q$.

Satisfiability in the relational calculus

Definition

A Boolean relational calculus query q is **satisfiable** if there exists a (finite) database D such that $D \models q$.

Theorem ([Trakhtenbrot, 1963])

*Satisfiability of the relational calculus (in the finite case) is **undecidable** and **recursively enumerable**.*

Satisfiability in the relational calculus

Definition

A Boolean relational calculus query q is **satisfiable** if there exists a (finite) database D such that $D \models q$.

Theorem ([Trakhtenbrot, 1963])

*Satisfiability of the relational calculus (in the finite case) is **undecidable** and **recursively enumerable**.*

Proof.

Hardness: Reduction from the POST correspondence problem, technical, see [Abiteboul et al., 1995].

R.-E.: enumerate all databases. □

Isn't it the same as undecidability of FO satisfiability?

Isn't it the same as undecidability of FO satisfiability?

Theorem

*Satisfiability of first-order logic over infinite models is
undecidable and co-recursively enumerable.*

Isn't it the same as undecidability of FO satisfiability?

Theorem

*Satisfiability of first-order logic over infinite models is **undecidable** and **co-recursively enumerable**.*

Proof.

Hardness: Code the **non-halting of a Turing machine** in a first-order logic formula.

Co-R.-E.: Thanks to **Gödel's completeness theorem**, every valid formula (i.e., every formula whose negation is unsatisfiable) has a proof, just enumerate these proofs. □

Isn't it the same as undecidability of FO satisfiability?

Theorem

*Satisfiability of first-order logic over infinite models is **undecidable** and **co-recursively enumerable**.*

Proof.

Hardness: Code the **non-halting of a Turing machine** in a first-order logic formula.

Co-R.-E.: Thanks to **Gödel's completeness theorem**, every valid formula (i.e., every formula whose negation is unsatisfiable) has a proof, just enumerate these proofs. □

Co-R.E. vs R.E.! No equivalent of Gödel's completeness theorem for finite models!

Containment and equivalence of the calculus

Theorem

*Containment and equivalence of relational calculus queries are **undecidable** and **co-recursively enumerable**.*

Containment and equivalence of the calculus

Theorem

*Containment and equivalence of relational calculus queries are **undecidable** and **co-recursively enumerable**.*

Proof.

Undecidability is by direct reduction from the undecidability of satisfiability.

Co-recursive enumerability is shown directly, by enumerating possible counter-examples. □

Plan

Relational Calculus

Complexity of Query Evaluation

Static Analysis of Queries

Conclusion

Conclusions

- Relational calculus: A logical query language equivalent to the relational algebra (or to the core of SQL)
- Query evaluation is very efficient!
- Does not capture everything polynomial-time: need for recursive query languages
- Undecidable static analysis: restrict to conjunctive queries for optimization (cf. next lecture)

Bibliography I

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

<http://webdam.inria.fr/Alice/>.

Edgar F. Codd. Relational completeness of data base sublanguages. In *Data Base Systems. Courant Computer Science Symposium*, 1972.

Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi: 10.1007/978-3-662-07003-1. URL

<http://dx.doi.org/10.1007/978-3-662-07003-1>.

Boris A. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *American Mathematical Society Translations Series 2*, 23:1–5, 1963.