Summary: The Complexity of Relational Query Languages

s1856331

February 2019

1 Introduction

Relational databases as introduced by Edgar Codd store data in relations. This data is retrieved by query based on logical and algebraic languages. Logical languages are formulas that return tuples that satisfy them over a database while algebraic languages express the order of operations of a query over a given database. Alfredo et al. [1] identify two properties that should be satisfied by a relational query language, "that the value produced by a query should be independent of the manner in which the data are actually stored in the database" and secondly, "that a query language should treat data values as essentially uninterpreted objects..."

To evaluate the efficiency of query execution, the paper by Vardi et al. [12] studies two measures of complexity for query languages, data complexity and expression complexity. Data complexity is the complexity of evaluating the query of a language as a function of the size of the database. It can be viewed as how difficult it is to query individual results. Expression complexity evaluates the brevity of a language over different datasets; an evaluation of a query as a function of the size of the expression that defines the query. Combined complexity, which is applying arbitrary expressions of a language to arbitrary databases is not explored because it is close to expression complexity. A highlight is that expression complexity is one exponential higher than data complexity; the intuition being that characteristics and succinctness of queries matter more than the size of the database. Specifically contrasted is the data complexity of languages being in LOGSPACE, NLOGSPACE, PTIME, NPTIME and PSPACE while their expression complexities being complete in PSPACE, NPSPACE, EXPTIME, NEXPTIME and EXPSPACE respectively.

The paper studies data and expression complexities of logical languages by restricting them from using quantifiers, implementing transitive closures, fixed points and second order existential quantification. For algebraic languages, it studies their extensions by bounded and unbounded looping as well as restricting them from projection.

The implication of restricting first-order logic languages from the use of quantifiers can be envisaged by taking a language with s characters and a sentence φ (a sentence is a Boolean query returning true or false) evaluating φ on a database of size n. The cycle is at most n^s for mapping all free variables (due to lack of quantifiers) in φ to the database. This mapping is accomplishable in *O(logn)* space. This implies that every database that satisfies φ is in LOGSPACE. In contrast, Chandra [3] denotes that if the database on which the query φ was ran on was neither empty nor did it have all tuples with complete information (a non-trivial database) then the set of all first-order sentences like φ would be PSPACE-complete as the query is considered as a general query.

2 Database Queries and Complexity

Definitions of query complexity

Definition 1: A relational database is a tuple $B=(D,R_1,...,R_k)$ where $D \subseteq N$ is a finite set and each i in $1 \leq i \leq k$, $R_i \subseteq D^{ai}$ for some a≥0. N denotes the set of natural numbers and the number a_i is the rank of R_i. Database B is considered of type $\bar{a}=(a_1,...,a_k)$.

Definition 2: A query of type $\bar{a} \rightarrow b$is a partial function *Q:B—B is of type* $\bar{a} \rightarrow 2^{Nb}$ such that if $B=(D,\bar{R})$ and $Q(B)$ is defined, then the query $Q(B) \subseteq D^b$.

Definition 3: A query language is a set of expressions of L and a meaning function μ such that for every expression e in L, $\mu(e)$ is a query. $C(L)$ denotes the class of queries defined by expressions in L such as; $C(L)= \{Q \mid Q=\mu(e) \textit{ for some } e \in L\}$. Q_e is used in place of $\mu(e)$ when μ is understood from the context. Given that all queries are functions, the paper takes the approach of measuring complexity as a recognition problem rather than a computation one. Intuitively, studying the difficulty of recognizing whether a tuple should appear in the result of running a query in a database.

Definition 4: The graph query Q is the set $Gr(Q)=\{(\bar{d},B) \mid \bar{d} \in Q(B)\}$ while the graph of a database B with respect to a language L is the set denoted as $Gr\{L\}(B)=\{(\bar{d},e) \mid e \in L \textit{ and } \bar{d} \in Q_e(B)\}$. $\bar{d}$ is the sub-domain of the database bounded by the query. The data complexity of a language L is the complexity of the sets $Gr(Q_e)$ for all expressions e in L. expression complexity of a language L is the complexity of the sets $Gr_L(B)$.

Definition 5: A language L is data-complete(denoted as D-complete) in a complexity class C if for every expression e in L, the data complexity $Gr(Q_e)$ is in C and there is an expression e_0 in L for which all sets in C are logspace reducible to the data complexity of the expression $Gr(Qe_0)$. L is expression-complete (E-complete) in a complexity class C if for every database B, the expression complexity $Gr_L(B)$ is in C and there is a database B_0 such that all the sets in C are logspace reducible to $Gr_L(B_0)$.

Definition 6: C being a complexity class, QC is the class of queries whose graph is in C, i.e., $QC=\{Q \mid Gr(Q) \in C\}$.

3 Logical Languages and their Complexities

First-order logic

3.0.1 Language 1

Let L be a first-order language with equality, no function symbols and with R_1,R_2,... as its predicate symbols. (R_i has been used to denote the formal definition and as the relation itself. The rank of R_i is a_i). Let L be the language consisting of expressions of the form $\bar{x}\varphi\bar{x}$ where φ is a formula of L and $\bar{x}$ is a distinct variable vector containing all free variables of φ. If e is such an expression, $\mu(\bar{e})$ is a query Q_e of type (a_1,...,a_k) $\rightarrow |\bar{x}|$. Query Q_e is defined by $Q_e(D,\bar{R}) = \{\bar{d} \in D^x \mid \varphi(\bar{d}) \textit{ is true in } (D,\bar{R})$. Let F- denote the quantifier-free version of F such that quantifiers are not allowed in F-. It is evident that F is more expressive than F-.

Example 1.1: Let R_1 be a binary relation describing friends on a social networking platform such that $R_1(x,y)$ means a user x of the network is friends with user y. The expression $(x,y).\exists z(R_1(x,z) \wedge R_2(z,y))$ represents the query of type (2) → 2, returning the concatenation of relation R_1 with itself. This describes 2 users of the network that are connected through a mutual friend. As illustrated by Aho [1], language F cannot express a query that returns pairs of users who are connected by an unbounded positive number of mutual friends. Aho argues on the importance of developing a "method for evaluating selections ahead of LPF (least fixed point) operators [thus converting] general transitive closure problem[s] into a single source shortest path problem". This guides on enriching F with transitive closure.

3.0.2 Language 2

Let TF be the language obtained by adding to F expressions of the form $T.(x,y).\varphi(x,y)$ where $(x,y).\varphi(x,y)$ is an expression of F. If e is such an expression then $\mu(e)$ is a query Q_e of type $(\bar{a})$

$\rightarrow$ 2 defined by $Q_e(D,\vec{R})=(Q_{(x,y)}.\varphi(D,\vec{R}))^+$. $^+$ denotes the transitive closure.

Example 1.2: Expressing a query that returns pairs of users who are connected by an unbounded positive number of mutual friends is be represented by *T.(x,y).*φ*(x,y).* Let R$_2$ be a ternary relation describing a group in the social network; i.e $R_2(x,y,z)$ with x being a group, y being a user in the group and z being another user in the group who is a friend to y. A query that returns friend suggestions to user x based on friends of y who are in the same group as x. This query cannot be represented in *TF*, necessitating enriching *F* by the fixpoint construct.

3.0.3 Language 3

Let *YF* be the language obtained by adding to *F* expressions of the form $YR.\bar{x}.\varphi(\bar{x})$, where $\bar{x}.\varphi(\bar{x})$ is an expression of *F* and *R* is a predicate symbol of rank |$\vec{\mathrm{x}}$| that occurs positively in φ such that each occurrence of *R* in is under an even number of negations. If e is such an expression then $\mu(e)$ is a query Q_e of type $(\bar{a}) \rightarrow |\vec{x}|$. $Q_e(D,\vec{R})$ is a relation R such that $Q_x,\varphi(D,R,\vec{R})$=R and for any relation R', if $Q_x,\varphi(D,R',\vec{R})$=R' then R$\subseteq$R'(least fixpoint). As shown by Chandra et al. [2], all the expressions in *YF* define total queries. If user x has friends y and v, consider a query that only returns friend suggestions to user x for only users who appear in both user y and v's friends list. This requires second-order existential quantification, a feature not in language YF.

3.0.4 Language 4

Let *SF* be the language obtained by adding to *F* expressions of the form $\bar{x}.\exists R.\varphi(\bar{x})$, where $\bar{x}.\varphi(\bar{x})$ is an expression of *F*. If e is such an expression and *R* is of arity a then μ(e) is a query Q_e of type $(\bar{a}) \rightarrow |\vec{x}|$.Queries defined by expressions in *YF* can be defined by using second-order universal and existential quantification. This shows that *SF* is more expressive than *YF*.

In the order of expressiveness, the defined languages can be illustrated as *Q(F-)*$\subset$*Q(F)*$\subset$*Q(TF)*$\subset$ *Q(YF)*$\subset$*Q(SF).*

3.1 Data and Expression Complexity

Fix database B_0*=(*D_0*,*R_0*), where* D_0*=*$\{0,1\}$ *and* R_0*=*$\{1\}$.

Theorem 1: F- is D-complete and E-complete in LOGSPACE. Proof: Nancy [10] proofs that evaluation of Boolean sentences is log space computable. For $e\in F$-, $Gr(Q_e)$ is in LOGSPACE because *F-* is a Boolean query.

Theorem 2: F is D-complete in LOGSPACE and E-complete in PSPACE. Proof: An algorithm iterates through all possible substitutions for quantified variables. The algorithm has logarithmic space data complexity and polynomial space expression complexity. D-completeness is trivially seen from the nature of the algorithm. Hartmanis [5] suggests a finetune of the data complexity by using a two-way deterministic finite automaton *(2DFA)* with k heads. E-completeness is proved by reduction from Stockmeyer's [11] Quantified Boolean Formulas. $Gr_F(B_0)$ is PSPACE hard.

Theorem 3: Let $e\in F$; then $Gr(Q_e)\in 2DFA(qn(e)+2)$. Proof: The number of heads needed to move over the domain is *qn(e)*, giving an assignment of elements to the *qn(e)* quantified variables. This requires an additional head to move over the assignment of elements to the free variables while the last head moves over the relations. Interaction between the first and last heads gives the truth values to the atomic formulae in e. From Ibarra [6],for all $k\in N$, *2dfa(k)* is properly contained in *2dfa(k+2)* and thus $C(\{e \mid e\in F \textit{ and } qn(e)=k\})$ is properly contained in $C(\{e \mid e\in F \textit{ and } qn(e)=k+2\})$.

Theorem 4: *TF* is D-complete in NLOGSPACE and E-complete in PSPACE. Proof: We guess a sequence $a{=}a_1,a_2,...,a_k$ and test whether $(a_i,a_{i+1}){\in}Q_{(x,y).\varphi}(B)$ for $1{\le}i{\le}k$. The algorithm has a nondeterministic logarithmic space data complexity and a nondeterministic polynomial space expression complexity. D-completeness is NLOGSPACE-hard by reduction from the Graph Accessibility problem. E-completeness: From the fact that PSPACE=NPSPACE and $F{\subseteq}TF$.

Theorem 5: YF is D-complete in PTIME and E-complete in EXPTIME. Let the fixpoint expression e be $YR.\bar{x}.\varphi(\bar{x})$, where R is of rank $\bar{x}$. The algorithm used by Chandra [2] to evaluate $Qe(D,\bar{R})$ builds a sequence of increasing approximations for R starting with the empty relation. Since the cardinality of $Q_e(D,\bar{R})$ is bounded by the size of x, the length of the sequence has the same bound. Therefore, the algorithm has a polynomial time data complexity and an exponential time expression complexity. D-Completeness: Derived from reduction from Jones' [7] Path System Accessibility problem. E-Completeness: An algorithm that evaluates $Q_e(D,\bar{R})$ making at most D^x iterations with the cost of being exponential asymptotically.

Theorem 6: There is a polynomial p such that for every $k{\in}N$, there is an expression $e{=}YR.\bar{x}.\varphi(\bar{x})$ with the size $\bar{x}{=}O(k)$ and $||\varphi||{=}O(p(k))$, such that $Gr(Q_e){\notin}DTIME(n^{k-1})$. Proof: There is a polynomial p so that given a deterministic Turing machine M with input alphabet Σ that operates in time $O(n^k)$, we can build an expression e with $|\bar{x}| = O(k)$ and $||\varphi||$=O(p(||M||)), such that, for every $s{\in}\Sigma$, one can construct in time $O(n^k)$ a database B and a vector $\bar{d}$.

Theorem 7: SF is D-complete in NPTIME and E-complete in NEXPTIME. Proof: A nondeterministic polynomial time data complexity and nondeterministic exponential time expression algorithm performs a guess and check procedure. D-completeness is achieved by reducing form Karp's [8] Clique problem. E-completeness: Any set accepted by a nondeterministic exponential time Turing machine is reducible to $Gr_{SF}(B_0)$ by using the encoding described in theorem 5.

4 Algebraic Languages and their Complexity

Relational algebra The language consists of variables and terms. If term t_1,t_2,t are terms, then the following are also terms: t1×t2 - Cartesian product, t1∪t2 - union, ¬t - negation, Proj$_i$(t) - projection of column i, Perm$_\theta$(t) - permutation by θ and Restrict$_{i,j}$(t) - restriction of columns i and j.

Definition 7: The language A consists of statements of the form: ${X_i}^a{\leftarrow}t$ where t is a term of rank a, $(S_1;S_2)$ where S_1 and S_2 are statements. The language *A-* is the projection-free version of *A*. The language *BA* consists of the statements of *A* and for $|t|$ do *S*, where t is a term and *S* is a statement. The language *LA* consists of the statements of *BA* and also while *t= do S* where t is a term and *S* is a statement.An expression in any of these languages is a pair *(S,t)* where *S* is a statement and t is an expression in the language. The semantics of the language are standard relational where Variable ${X_i}^a{\leftarrow}t$ has rank a. All variables are initialized to . The values of terms D, R_i are from the database. As is with first order logic, *A* is more expressive than *A*-, *BA* is more expressive than *A*, and *LA* is at least as expressive as *BA*. It is unclear if *LA* is more expressive than *BA*. Illustratively, $C(A\text{-}){\subset}C(A){\subset}C(BA){\subseteq}C(LA)$.

Theorem 8: *A-* is D-Complete in LOGSPACE and E-complete in PTIME. Proof: E-completeness in PTIME is derived from Ladner's [9] Circuit Value problem.

Theorem 9: A is D-complete in LOGSPACE and E-Complete in PSPACE. Proof: For every $e{\in}F$ there exists $e_2{\in}A$ such that $Q_{e1}{=}Q_{e2}$, and for every $e_2{\in}A$ there exists $e_1{\in}F$ such that $Q_{31}{=}Q_{e2}$, as described by Codd [4]. Additionally, translation can the performed in logarithmic space in both directions. This theorem proves theorem 8 as well.

Theorem 10: BA is D-complete in PTIME and E-complete in EXPTIME. Proof: Let e=(S,t)BA. The maximal rank of a term in e is bounded by $||e||$. Therefore the cardinality of all terms in e is bounded by the size of the database exponentiated to the square of the length of e. This shows a polynomial time data complexity and an exponential time expression complexity.

Theorem 11: LA is D-complete in PSPACE and E-complete in EXPSPACE. Proof: Let $e=(S,t)\in LA$. The cardinality of all terms in e is bounded by The size of the database exponentiated to the length of e. Therefore the evaluation has a polynomial space data complexity and an exponential space expression complexity. D-completeness has been proved by Chandra [2]. E-completeness: Any set accepted by an exponential time deterministic Turing machine is logspace reducible to $Gr_{LA}(B_0)$ by encoding the configuration of length 2^n by a 2n+m-ary relation R; with m depending on the machine in question. $R(a_1,\ldots,a_n,a_{n+1},\ldots,a_{n+m})$ means that the tape has symbol σ in its *j-th* square, where $(a_1,\ldots,a_n)$ and $(a_{n+1},\ldots,a_{n+m})$ are binary encodings of j and σ respectively. An unbounded loop simulates the unbounded computation of the machine starting from the initial configuration.

5 Conclusion

Closing languages *TF*, *YF* and *SF* under logical connectives yields *T*, *Y* and *S* whose hierarchies of expressions and queries can be defined making it plausible to investigate their expressiveness and complexity. It has been shown that first-order hierarchy is infinite and E-complete, as in Stockmeyer's [11] polynomial hierarchy. Additionally, it can be shown that transitive closure hierarchy is D-complete in Chandra's [2] logarithmic hierarchy while second-order hierarchy is E-complete in an analogously defined exponential hierarchy.

References

[1] Alfred V Aho and Jeffrey D Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 110–119. ACM, 1979.

[2] Ashok Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and system Sciences*, 25(1):99–128, 1982.

[3] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[4] Edgar F Codd. *Relational completeness of data base sublanguages.* Citeseer, 1972.

[5] Juris Hartmanis. On non-determinancy in simple computing devices. *Acta Informatica*, 1(4):336–344, 1972.

[6] Oscar H Ibarra. On two-way multihead automata. *Journal of Computer and System Sciences*, 7(1):28–36, 1973.

[7] Neil D Jones and William T Laaser. Complete problems for deterministic polynomial time. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 40–46. ACM, 1974.

[8] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[9] Richard E Ladner. The circuit value problem is log space complete for p. *ACM Sigact News*, 7(1):18–20, 1975.

[10] Nancy Lynch. Log space recognition and translation of parenthesis languages. *Journal of the ACM (JACM)*, 24(4):583–590, 1977.

[11] Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[12] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.