CS 6320: Spring 2009
From Declarative Languages to Scalable Systems
Review

Guozhang Wang

March 22, 2009

Declarative language, such as SQL, has been proved successful in particular architecture due to its efficient processing algorithms and global optimizations. Currently it is also being used in many other fields like Games, Stream Processing, etc. Issues involved in a declarative language ddesign include complexity-expressiveness tradeoffs, algebraization and engine implementation, (heuristic, cost-based, global) optimization, problem decompositions, etc.

In the first section we discuss query engine, which is not only relevant to building databases but also to ground the later material in the physical reality.

1 Introduction: A Basic Query Processor

1.1 Cost of Secondary-storage Algorithms: Hard Disks

- Seek time: time to move the read/write head to the right position ($\approx 10ms$); Transfer time per page ($\approx 0.1ms$)
- Cost of execution of secondary-storage: $\#pageI/Os \times transfertime + \#seeks \times seektime$
- This I/O cost dominates CPU cost in most scenarios

1.2 Cost of Join Operators

Join is the most costly operator in query processing. Different join algorithms have different assumptions: Hash and Merge-Join assumes a 1:n relationship; Index Nested Loop Join assumes the partners in S of each R tuple fit into one page.

1.3 Relational Query Optimization

One main motivation for special-purpose query languages such as declarative SQL compared with general-purpose programming languages is query optimization. Since the cost can be re-estimated, it is more efficient for cost-based query optimization in addition to heuristic query optimization.

Based on the algebraic laws, some heuristics can be applied first to optimization to reduce intermediate result sizes: pushing selections down, pushing projections down, join reordering, etc.

After heuristic methods, further optimization can be made by firstly estimating sizes of (intermediate) results according to selectivity of operators. One note is that selection, projection, BNL join, and Index NL join can be pipelined instead of materialized.

Several observations: indexes are not always worth using, choose Index NL Join over BNL Join is there are very few tuples in the outer relation and the index is clustered. Join order is usually even more important than choice of operator, and also more difficult to optimize, since the former must be decided globally while the latter can be chosen individually, operator by operator: use dynamic programming to build the plan in a bottom-up way.

2 Conjunctive Queries in Depth

2.1 Complexity of conjunctive queries

Each query can be presented as the form of conjunctives of predicates, where the query result is the set of all mappings of tuples expressed in the "*head*" such that the mapping θ from each variables and constants in "*body*" appears in the database. Such a query can be expressed using a tableau.

The first complexity result of conjunctive query is: conjunctive query evaluation is $NP-complete$ (Chandra, Merlin), and the proof is by deriving from 3-colorability. However it is not saying that the query is impossible to evaluate, it means the query time will be increasing sharply by the number of predicates of the query, but somehow independent on the database size (which is the concern of the performance scalability).

2.2 Homomorphism Theorem

2.2.1 Containment and Equivalence

Intuitively it would be great to remove joins in the query while keeping the modified query semantically equivalent. The question is: how to define the equivalence between queries? *Query Containment* and $Homomorphism$ gives the solution. For formal definitions to these two concepts, refer to [1].

Theorem 2.1 (Chandra and Merlin 1977) *Let q = (T, u) and q' =*

(T', u') be tableau queries over the same schema R. Then, q is contained in q' iff there exists a homomorphism from q' to q.

Note the proof of the above theorem uses the composability property of homomorphism, which means $\theta_1(\theta_2(q)) = \theta_1 \circ \theta_2(q)$. Now given the theorem, evaluating a conjunctive query Q on a database I is the same problem as finding all homomorphisms from Q to I. Furthermore, checking the query containment of Q_1 in Q_2 can be done by firstly "freezing" atoms of body of Q_1 as a database instance and evaluate $Q2$ on this database. If head of Q_1 is in query result of Q_2 then containment holds and vise versa.

Theorem 2.2 *Given a CQ q = (T, u), there is a minimal equivalen CQ q' = (T', u') such that T' is a subset of T. Furthermore, if q and q' are both minimal and equivalent, then they are isomorphic.*

One note is that the three problems: query containment, query equivalence, and minimization is NP-complete for CQs and undecidable for the full language of FO queries. For the first statement, however, it is only based on the length of the query (# of tuples in the tableau.) which is normally short, thus a simple algorithm can be used. For instance of finding minimal homomorphism query, one can iterate all the sub-queries (exponential number) in a bottom-up manner and check its containment (NP-complete) using the above "freezing" algorithm. Since we search in the subquery space in a bottom-up manner of the query size, once one sub-query is found, algorithm terminates and outputs.

2.2.2 Dependencies

Some more features can be considered while trying to minimize the query. *Functional dependencies (fds) (indicating primary key constraint)* and *Inclusion dependencies (inds) (indicating foreign key constraint)* are two further constraints that can be used to shrink the query size.

Chase rules is a tool for reasoning with conjunctive queries and a large class of dependencies, and to minimize these queries under a set of dependencies. For each of the two dependency classes, it has an action rule. The containment relationship holds after chase operations on the queries. For details refer to the slides.

Although Chase operations may not terminate under arbitrary set of dependencies. We have two properties of it under certain dependency and query types.

- For fds and acyclic inds, containment is decidable.
- For a CQ and set of fds and acyclic inds, the chase terminates in exponential time.

2.3 Data Integration

2.3.1 Motivation

Since database schemas have been developed independently, problem of heterogeneity arises under the applications of data exchange, database merge operation, data warehousing, and distributed/P2P information systems. Therefore, data integration is needed to "unify" the different schemas.

The scenario is as follows: the input is several source databases with different schemas, a destination schema (although this "global schema" may not be pre-defined in real applications, they can be collected and made using some other IE techniques or through expert designs) and the semantic mappings from source databases to the global schema. The goal is to evaluate queries that are formulated using the destination schema outputing as many relevant results as possible given the source databases.

2.3.2 Global-as-view Integration

A simple integration method can be applied based on the assumption that the source relations represent data in a finer granularity level than the global schema. Then the global schema objects can be defined using joins, aggregations, etc. from the stored source data.

However, if the global schema is of finer granularity of the source data or the join operation is lossy due to semantic incompleteness this method can on longer be useful.

2.3.3 Local-as-view

To solve the problems of Global-as-view (GAV), Local-as-view (LAV) is proposed where the source relations are "defined" as views. And when a query on global schemas is issued, we try to answer it *only* using the materialized views.

Materialized views cannot be implemented under "lazy evaluation", when the views are only composed with the query at evaluation time. Instead we need to apply "eager evaluation", which materialize views beforehand as pre-computed partial results of the query. Note the mapping is from global schema on which query is issued to the source database as views, while data is from the source database as views to the query schema.

In some cases, if we *only* use the views to answer query then the result is not equal (for example, using view "grandparent" to find all great-grandparent). The notion of *Maximally Contained Rewriting* (MCR) is proposed to make the returned result using rewritten queries as close as possible to the result returned by the original queries. Whether a set of MCRs is "equal" to the original query can be checked by homomorphism.

Since without the presence of fds, no MCR needs to have more body atom than the original query. The search space for MCRs can be restricted. One note is that this MCR definition is under the *Open World Assumption*, which assumes the source relation stores only a subset tuples of the view on the "real world" (global schema). If we use *Closed World Assumption*, then all the possible "real worlds" can be determined and the "certain answers" can be derived to achieve more complete results (CWA relax the constraints on rewriting and hence make more rewriting queries possible). However, query evaluation under CWA is more complex than under the OWA.

2.3.4 Inverse Rules and GLAV Mapping

Now we need to answer the question: given the (G)LAV mapping and the query, how can we find the MCR that only uses the source database as views. The solution is to use *Inverse Rules*, which works as follows:

- For the give LAV or GAV formed in *Source-to-target-dependencies*, construct the inverse rules for each atom of the "target".
- Try all possible *compositions* of the inverse rules with the query. For each compositions of the inverse rules with the query, compute the *unification* which returns either a unified rewriting query or "failure".
- if the current composition of rules results in a "failure" we simply drop the generated rewriting query. For those success rewriting queries we choose the maximum contained rewriting query.

This algorithm guarantees that the result query is the maximum contained rewriting query with certain answer tuples under Open World Assumption. That is, we maximize the "recall" under the constraint of perfect precision of OWA. After we get the rewriting query, we evaluate on the views as follows:

- Evaluate the inverse rules on view V. That is, for each tuple in V, generates its possible tuples in "imagined original database" R using the inverse rules. Then if we have n rules and m tuples in V, there will be mn possible tuples in R.
- Turn all the function term into variables (like "labeled nulls").
- Compute the minimized version $core(R)$ of R using homomorphism.
- Enforce target dependencies using Chase if any.
- Execute the rewriting query on $core(R)$.
- Drop all tuples with variables in it since we only need *certain answers*.

Now what we get is the certain answer given the view and the open world assumption. Data exchange, which uses this way to map data between different databases, can be treated as a subproblem of data integration.

2.4 Structural decomposition

2.4.1 GYO Algorithm

A hypergraph is a generalized graph when edges could link more than two nodes. When each edge only links two nodes the hypergraph is reduced to a normal graph. A hypergraph can be used to present conjunctive queries as follows: each attribute appeared in the query is a node, each atom in the predicate is an edge linking all the nodes presented by attributes in the atom.

We are interested in some of the query transformed hypergraph's properties to decompose queries. One is the acyclicity. We can check this property using *GYO* algorithm.

Definition 2.3 *A hypergraph is call acyclic iff its GYO-reduced graph (which is the output of GYO algorithm) is* empty*.*

The GYO algorithm executes by iteratively choose an *ear* of the graph and deletes it, under it cannot find any more ears and stop. If a query's corresponding hypergraph is proved acyclic by GYO algorithm, it is clear that we can derive a efficient query plan for this query from the hypertree. However, if the hypergraph is cyclic, what can we do? We answer this question in the next section.

2.5 Hypertree Decomposition

Before we present Hypertree decomposition to find efficient query plan, we firstly focus on a special class of queries: Boolean Conjunctive Query (BCQ). A BCQ query asks whether the relations has an nonempty result for the conjunctive query, and hence is equal to a constraint satisfaction problem (CSP) in AI (eg, cross world puzzle). We have the following complexity results for BCQ:

- Evaluating BCQ is NP-complete in the general case; furthermore, it is NP-hard even for fixed database. That is, if n is size of the database, m is the number of atoms in the query, then it requires worst-case complexity: $O(nm)$.
- However, it has polynomial algorithm if query has a nearly acyclic hypergraph.

Now we explain what does it mean by "nearly acyclic" by *tree width*. Tree width is a measure of the cyclicity of graphs: a real tree has tree width of 2, and as the width of the graph increases, the graph is less and less "like" a tree. A graph has tree width k if we can compute a tree decomposition with tree width k (note if a graph has tree width k then it has tree width k' for any

$k' \geq k$). One reason to use tree width as cyclicity measure is that checking if a graph has tree width k by computing a tree decomposition is linear time. For a detailed definition one can refer to [2]. BCQ is polynomial for queries of bounded tree width (since the exponential factor is transformed to tree width k from n), if a tree decomposition on its *primal tree* is given.

Given the tree decomposition we can derive an efficient query decomposition (or query plan). If the tree decomposition has tree width 1, then we can simply take the tree decomposition as the query plan, with each intermediate result of the child node treated as a filter to the parent node condition result. For query decomposition of tree width more than 1, we derive the query plan by constructing corresponding predicates which contains the atoms of its decomposed tree nodes. Besides, we have an important observation: Query atoms can be used and reused "partially" as long as the full atom appears somewhere else. This can help us achieve a more efficient query decomposition than the original tree decomposition by reusing some predicates and projecting some atoms in it.

3 First Order Queries

3.1 Introduction to First Order Logic

The first order logic is composed by *terms*(contains *variables, constants*), *relation names* and $\forall, \exists, \neg, \wedge, \vee$. For definitions of *free variables, quantifier rank, structure (database), signature (signature)* please refer to [1]. The semantics of FO is given by *satisfaction relation* $\models$, which denotes that structure satisfies formula when the free variables of the formula are replaced by the values of the structure everywhere in the formula. Note that the equivalence of formulae can be utilized to evaluate the query.

3.2 Relational Domain Calculus

When ϕ is an FO formula with $free(\phi) = x_1, ..., x_n$, then ¡$x_1, ..., x_n$¿—$\phi$ is an n-ary query of the domain calculus. Domain calculus is used to write (or translate) a algebra/SQL query given an informal problem statement. There are several steps to write a very complicated queries without mistakes:

1. Turn the problem statement in English into fully precisely "quantified English" by introducing variables to name objects and making the quantifiers explicit.

2. Turn the quantified English query into a calculus query (adding *active domain relation*).

3. Turn the calculus query into algebra/SQL query.

One note for translating into "quantified English" is *natural language ambiguity*, which may cause difficulties in queries with "all", "only", "maxima", "comparing sets". In next section we show how to execute the second and third steps.

3.3 Domain Independence and Codd's Theorem

Not all quantified English can be translated into domain calculus, some queries are "unsafe" since they do not hold the *domain independence* properties and must be avoided. Domain independence is an undecidable property for FO queries, thus we use *range restriction* as a sufficient syntactic condition to check domain independence.

For every domain-independent FO query, there is an equivalent range-restricted FO query. We can check range-restriction in $SRNF$ form of the FO formula. Given range-restricted FO formula we have the following important theorem:

Theorem 3.1 (Codd) *A query is definable in relational algebra if and only if it is definable in the range-restricted domain calculus.*

The proof from algebra to FO is straightforward by definition. The proof from FO to algebra is 1) put FO into SRNF; 2) put SRNF into Relational-Algebra NF (RANF), which requires range-restriction for each "subformula"; 3) translate RANF into relational algebra.

4 EF Games for Inexpressibility Proofs

4.1 Methodology Theorem

Although there is a standard technique for inexpressibility proofs from logic called *compactness*, it will fail in finite structures (databases). Therefore we need another technique to prove such inexpressibility, and *EF games* are such a technique.

There are two players: Spoiler S and Duplicator D, and two structures of the *same* schema. The number of moves k to be played is fixed in advance. Spoiler starts first and may choose its structure anew in each move. Duplicator always has to answer in the other structure. The task of the spoiler is to "destroy" the *Partial Isomorphism* based on the mappings between elements of the two structures defined by the game run, and the task of the duplicator is to "hold" it.

We can prove that there is either a winning strategy for the spoiler or a winning strategy for the duplicator (since for spoiler the winning strategy is $thereis..suchthatforall..thereis..suchthatforall..$, and for duplicator the winning strategy is $forall..thereis..suchthatforall..$ and this two statement are complementary). If the spoiler has a winning strategy, then there is an

FO sentence that distinguishes between the two structure (note these two structures can be finite). The construction algorithm of the FO is that, for a winning strategy for spoiler as the input:

- Construct a sentence ϕ which is true on the structure on which spoiler puts the first token and is false on the other.
- Spoiler's choice of structure in move i decides the $i-th$ quantifier (note there may be switches of current structure).
- The alternative answer of duplicator are combined using conjunctions.

Given the above conclusion, we have the following main theorem:

Theorem 4.1 (Ehrenfeucht, Fraïssé) *Given two structures A and B and an integer k. Then the following statements are equivalent:*

1. *A and B cannot be distinguished by FO sentences of quantifier rank k.*
2. *Duplicator has a winning strategy for the k-move EF game.*

We have proved that $1 \Rightarrow 2$, we still have to prove the other direction. The idea is to construct a winning strategy for spoiler for the $k-move$ EF game for a formula ϕ of quantifier rank k with $A \vDash \phi$ and $B \vDash \neg\phi$. For details refer to [1].

Since expressibility of a query in FO means just that there is an FO formula equivalent to that query, and if there is such a formula, it must have some quantifier rank, given the above theorem, we get the following Methodology theorem:

Theorem 4.2 (Methodology theorem) *Given a Boolean query Q. There is* ***no*** *FO sentence that expresses Q if and only if there are, for each k, structures A_k, B_k such that:*

1. $A_k \vDash \phi$.
2. $B_k \vDash \neg\phi$.
3. $A_k \sim_k B_k$.

Therefore to prove inexpressibility, we only have to 1) construct suitable structures A_k and B_k for any k and 2) prove that $A_k \sim_k B_k$. By using the *Composition theorem for paths*, we can prove that a lot of queries such as parity query, Eulerian graph query, undirected paths, k-colorability, acyclicity and graph reachability are not expressible in FO.

From the viewpoint of Algebraic, this theorem guarantees the *forth and back property* of a sequence of partial isomorphisms between two structures.

4.2 0-1-Laws

To prove using EF games that a query Q is not expressible in FO, we have to design a family of pairs of structures $< A_k, B_k >$ for any $k \geq 1$, that the duplicator has a winning strategy but the query has different answers on these two structures. On the other hand, a simpler proof technique can be used, which is called *0-1-Laws*.

Given a query, we can define *almost certainly true/false* based on the domain size ratio (note some queries do not have limits so they are neither almost certainly true or almost certainly false).

Definition 4.3 *Given a query language L (FO, SQL, English, . . .), if each Boolean query $Q \in L$ is either a.c.t. or a.c.f., then we say that L has a 0-1 law. If a query is neither a.c.t nor a.c.f., then this query is not expressible under this language L.*

Lemma 4.1 *If the sentences $\phi_1, \phi_2, ..., \phi_m$ are a.c.t then $(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_m)$ is a.c.t.*

Theorem 4.4 *FO (without constants) has a 0-1 law.*

This theorem is proved using the lemma that each *extension axiom (EA)* is a.c.t., and that there exists a (unique up to isomorphism) countably infinite graph G such that $G \models EA$ where EA stands for all extension axioms (for any k), this graph is called *Rado graph*. The extension axiom FO sentence guarantees that for any moves that spoiler chooses in the EF game, the duplicator can always find a winning step (which means he has a winning strategy), if the structure satisfies all extension axioms. Thus for an arbitrary FO sentence ϕ: if it is true on the Rado graph or if it true on EA, it is a.c.t., and vice versa. Refer to [1] for the proof of this important theory.

4.3 Data and Query Complexity

4.3.1 Turing Machine

A complexity class is a set of problems, and many complexity classes are defined using Turing machines recognizing the language version of the *prototypical decision problem* of the complexity class.

There are two types of Turing machines: deterministic and non-deterministic, at each step, a deterministic TM will has only one way to go based on the current state and the symbol read by the header, while a non-deterministic TM could randomly choose one of the three options. A instantaneous description (id) is a state of the machine at a point in time. A run of TM is a sequence of ids, an accepted run is the run that ends at the accepted state.

Some important classes: $NTIME$, $NSPACE$, $DTIME$, $DSPACE$. Especially, $P := DTIME(n^{O(1)}), NP := NTIME(n^{O(1)}), PSPACE := DSPACE(n^{O(1)}), LOGSPACE := DSPACE(O^{(logn)})$, etc.

4.3.2 Complexity Measure for Queries

We n seen that conjunctive queries are NP-hard. Since FO queries (and thus SQL) strictly subsume them, FO queries are also NP-hard. However it is only *query complexity* of time, which is based on input query size. For space complexity, FO is PSPACE-complete with respect to query/combined complexity. Further, for a fixed query, the data complexity of FO is LOGSPACE.

However, we can use *Circuit Complexity* to give it an even better bound: the FO queries are in AC_0 w.r.t. data complexity. Since $NC_1 \subseteq LOGSPACE$, it is a better bound. Furthermore, if a query is in circuit complexity, it is implied to be highly paralleled. Thus the result says the query evaluation for FO can basically be arbitrarily scaled: given enough hardware, queries can be evaluated in constant time , no matter how large the database is. However, in practice, the circuit is too large to be materialized, thus this circuit parallel model must be modified before using.

5 Processing Queries using Automata

5.1 XML and XPath

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1995.

[2] J. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2005.