The expressive power of SQL

Submitted by :

Doris Fischer 011143252

Ayelet Kaidar 031756471

SQL vs. RA

- Aggregate functions - such as COUNT,AVG,MIN,MAX etc.
- Grouping – not only can one compute aggregates, one can also group them by values of different attributes.
- Arithmetic – SQL allows one to apply arithmetic operations to numerical values.

A simple example

- R(Src,Dest) is a relation with flight information, where Src stands for source and Dest stands for destination.
- To find pairs of cities (A,B) such that it's possible to fly from A to B with one stop, one would use a self- join:

A simple example (cont.)

- SQL:
  SELECT R1.Src,R2.Dest
  FROM R AS R1, R AS R2
  WHERE R1.Dest = R2.Src
- RA:
  \pai_{R1.Src,R2.Dest}(\sigma_{R1.Dest=R2.Src}(R1\JoinR2))

A simple example (cont.)

- Suppose we want pairs of cities such that one makes 2 stops on the way?
- SELECT R1.Src , R3.Dest

  FROM R AS R1, R AS R2 , R AS R3

  WHERE R1.Dest = R2.Src AND R2.Dest = R3.Src

A simple example (cont.)

- RA: \pai_{R1.Src,R3.Dest}(\sigma_{R1.Dest=R2.Src\andR2.Dest=R3.Src}(R1\JoinR2\JoinR3))

Transitive closure

- But often one needs a general reachability query in which no a priori bound on the number of stops in known; that is, whether it possible to get to B from A.
- Graph–theoretically,This means computing the transitive closure of R.
- It’s well known that the transitive closure of a graph is not expressible in relational algebra (RA) or calculus (RC)

Transitive closure (Cont.)

- In SQL3, one could write the reachability query as:
- WITH RECURSIVE TrCL(Src,Dest) AS

```
        R
UNION
        SELECT TrCL.Src , R.Dest
        FROM TrCL, R
        WHERE TrCL.Dest = R.Src
        SELECT * FROM TrCL
```

Transitive closure (Cont.)

- This simply models the usual datalog rules for transitive closure :
- trcl(x,y) :- r(x,y)

  trcl(x,y) :- trcl(x,z),r(z,y)

- The reason for introducing recursion in this SQL standard is precisely this folk result stating that it cannot be expressed in the language.
- But when one looks at what evidence is provided to support this claim, one notices that all the references point to papers , in which it proved that RA and RC cannot express recursive queries.
- Why is this not sufficient?

Consider this query

- SELECT R1.A
  FROM R1, R2
  WHERE(SELECT COUNT(*) FROM R1) >
  (SELECT COUNT(*) FROM R2)
- This query tests if |R1|>|R2|: in that case it returns the A attributes of R1,otherwise it returns the empty set.

- However , logicians proved, long ago, that first-order logic,and thus RC, cannot compare cardinalities of relations, and yet we have a very simple SQL query doing precisely that.
- The conclusion, of course, is that SQL has more power than RA, and the main source of this additional power is its *aggregation* and *grouping* constructs, together with *arithmetic* operations on numerical attributes.

- But than one cannot say that the transitive closure query is not expressible in SQL simply because it is inexpressible in RC.

RA with aggregates

- In order to deal with aggregation we need to distinguish numerical and non-numerical columns.
- We define 2 types:

  b - non numerical type with domain Dom.

  n - numerical type with domain Num.

  Num \belongs N,Z,Q,R etc.

RA with aggregates (Cont.)

- The type of a relation is a string over {b,n}.

  For example:

  the type of S(Empl,Salary) is bn.
- For a type t, $t.i$ is the ith position in the string. The length of t is denoted by $\boxtimes t \boxtimes$.
- A database scheme is a collection of relation names R_i and their types t_i .
- We write R_i:t_i if the type of R_i is t_i .

RA with aggregates (Cont.)

- \Omega – a collection of functions and predicates on Num.
- \teta – a collection of aggregates.

Locality of queries

- A query can only look at a small portion of its input.
- If the input is a graph, “small” means a neighborhood of a small radius.

Example1 - Reachability

- Relation R contains edges of a directed graph.
- R:bb is a *purely relational* schema (contains only non-numerical types).
- Suppose e is an expression of type bb.
- Given a pair (a,b) \belongs R , and a number r > 0:
  N_r^R (a,b) = *r-neighborhood* of a,b in R, is the sub graph on the set of nodes in R whose distance from either a or b is at most r.

Example1 – Reachability (cont.)

- N_r^R(a,b) = { x | |x-a| <= r or |x-b| <= r }
- (a,b) \isomorphic_r^R (c,d) meaning the 2 neighborhoods

  N_r^R(a,b) and N_r^R(c,d) are isomorphic,
- Reachability is not local.

Example2 - RC

- RC expressions are all local.
- See proof in class lectures – winning series for the second player.

Locality of queries

- e is local if there is a number r , depending on e only , s.t.:

  (a,b) \isomorphic_r^R (c,d) ℳ (a,b) \belongs to e(R) iff (c,d) \belongs to e(R)

ALG_aggr(All,All)

- Relational algebra with grouping and aggregate functions.
- When we consider only pure relational scheme the only aggregate function will be COUNT.

Theorem 1 – Locality of SQL

Let e be a pure relational graph query in ALG_aggr(All,All), that is, an expression of type bb over the scheme of one symbol R:bb. Then e is local.

Conclusion so far…

- Reachability is not expressible in SQL over the base type of b, no matter what aggregate functions and arithmetic operations are available.
- Inexpressiblity of many other queries can be derived from this, for example, tests for graph connectivity and acyclicity.

Question

Question: Are expressions in SQL(pure relational)-RA local?

Answer: YES.

Next steps…

1. Introduce an *aggregate logic* $\mathcal{L}$_aggr, as an extension of FOL and show how ALG_aggr queries are translated into it.
2. Replace aggregate terms by *counting quantifiers*, thereby translating $\mathcal{L}$_aggr into a simpler logic $\mathcal{L}$_C.
3. Inductive proof of locality of $\mathcal{L}$_C formulae.

The logic $\mathcal{L}$_aggr

- The structures are relational databases over 2 base types with domains Dom and Num; that is vocabularies are just schemas.
- The logic is *two-sorted.*
- Dom will be referred as *first-sort* and Num as *second-sort.*

The logic $\mathcal{L}$_aggr(\omega,\teta)

- A SC structure D is a tuple <A,R_1^D … , >, where A is a finite subset of Dom, and R_i^D is a finite subset of

  \pai from j=1 to |t_j|dom_j(D)

  where dom_j(D) = A for t_*i.j* = b,

  and dom_j(D) = Num for t_i.*j* = n.

The logic $\mathcal{L}$_aggr(\omega,\teta) cont.

- A variable of sort i is a term of sort i, i =1,2..
- If R:t is in SC, and u is a tuple of terms of type t, then R(u) is a formula.
- Formulae are closed under Boolean connectives and quantification (respecting sorts). If x is a first-sort variable, \exists x is interpreted as \exists x \belongs A; if k is a second-sort variable \exists k is interpreted as \exists k \belongs Num.

The logic $\mathcal{L}$_aggr (\omega,\teta) cont.

- If P is an n-ary predicate in \omega and \tau_1… \tau_n are second-sort terms, then P(\tau_1… \tau_n) is a formula.
- If f is an n-ary function in \omega and \tau_1… \tau_n are second-sort terms, then f(\tau_1… \tau_n) is a second-sort term.

The logic $\mathcal{L}$_aggr(\omega,\teta) cont.

- If $\mathcal{F}$ is an aggregate in \teta, \phi(\bar{x},\bar{y}) is a formula and \tau((\bar{x},\bar{y}) a second-sort term, then \tau‘(x) =Aggr_$\mathcal{F}$\bar{y}.(\phi((\bar{x},\bar{y}) ,\tau((\bar{x},\bar{y})) is a second-sort term with free variables x.

Theorem 2

Let e:t be an expression of ALG_{aggr} (\omega,\teta). Then there is a formula \phi{_e}(x) of $\mathcal{L}$_{aggr} (\omega,\teta), with x of type t, such that for any SC-database D,

$$e(D) = \{a \mid D \models \phi_e(a) \}$$

Theorem 2 – proof

- Numerical selection:
  e‘ = \sigma[P]_{i1…il}(e) , P \belongs \omega.
  \phi_{e’}(\bar{x}) \defined \phi_{e}(\bar{x}) \and P(x_i1…x_ik).
- Function application:
  e‘ = Apply[f]_{i1…ik}(e), f:Num^k♦Num f \belongs \omega.
  \phi_{e’}(\bar{x},q) \defined \phi_{e}(\bar{x}) \and (q=f(x_i1…x_ik).

Theorem 2 – proof (cont.)

- Aggregation:
  e‘ = Aggr[i:$\mathcal{F}$](e).
  \phi_{e’}(\bar{x},q) \defined
  \phi_{e}(\bar{x}) \and
  (q=Aggr_F\bar{y}.(\phi_{e}(\ bar{y}),y_i)).

Theorem 2 – proof (cont.)

- Grouping:
  e‘ = Group_m[\lamda S.e_1](e_2), where e_1:u is over
  SC \conjunction {S}, and e_2:ts is over SC.
  Let x:t,y:s,z:u. Then:
  \phi_{e’}(\bar{x},\bar{z}) \defined \exists \bar{y} \phi_{e_2}(\bar{x},\bar{y}) \and \phi_{e_1}(\bar{z})[\phi_{e_2}(\bar{x},\bar{v})/S(\bar{v})]

Theorem 3 (general)

Let e be a pure relational query in ALG_aggr(All,All), that is, an expression of type b….b over a pure relational schema.

Then e is local.

Translating L_aggr into L_c

- The logic L_c is simpler than L_aggr because it does not have aggregate terms.
- The price for the above will be that L_c has infinitary conjunctions and disjunctions.
- However, the translation ensures that for each infinite conjunction or disjunction, there is a uniform bound on the rank (Will be discussed later…) of formulae in it, and this property suffices to establish locality.

The logic L_c

- The structures are the same as for L_aggr.
- The only terms are variables (of any sort).
- Any constant c \belongs Num is a term of the second sort.
- Atomic formulae are R(x), where R \belongs SC, and x is a tuple of terms of the appropriate sort, and x \equals y, where x,y are terms of the same sort.
- Formulae are closed under Boolean connectives.

The logic L_C – cont.

- Formulae are infinitary connectives:
  if \phi_i , i \belongs I, is a collection of formulae, then
  \or i \belongs I \phi_i and \and I \belongs I \phi_i are L_C formulae.
- They are also closed under first and second sort quantification.

The logic L_C – cont.

- For every i \belongs N, there is a quantifier \exists i that binds one first sort variable: that is, if phi(x,\bar{y}) is a formula, then \exists ix \phi(x,\bar{y}) is a formula whose free variables are \bar{y}.

The logic L_C – cont.

- The semantic will be:
  D |= \exists ix \phi(x,\bar{a}) if there are i distinct elements b_1… b_i \belongs A s.t.
  D |= \exists ix \phi(b_i,\bar{a}) , 1<= j<= i.
  That is, the existential quantifier is witnessed by at least i elements.
- Note: \exists x\phi is equivalent to \exists 1x\phi.

The rank of a formula – L_C

- For L_C this is the quantifier rank, but the second sort quantification does not count.
- For each atomic \phi , rk(\phi) = 0.
- For \phi= \or_i \phi ,rk(\phi) = sup_i rk(\phi), and likewise for \and.
- rk(\not \phi) = rk(\phi).
- rk(\exists ix \phi) = rk(\phi)+1 for x first sort; rk(\exists k \phi) = rk(\phi) for k second sort.

The rank of a formula – L_aggr

- For L_aggr the definition differs slightly.
- For a variable or a constant term, the rank is 0.
- The rank of an atomic formula is the maximum rank of a term in it.
- rk(\phi_1* \phi_2)=max (rk((\phi_1),rk((\phi_2)), for *\belongs{\or,\and}
- rk(\not \phi) = rk(\phi).

The rank of a formula – L_aggr – cont.

- rk(f(\tau_1…\tau_n)) = max _{1<=i<=n}rk(\tau_i).
- rk(\exists x\phi) = rk(\phi)+1 if x is first sort; rk(\exists k \phi) = rk(\phi) if k is second sort.
- rk($\mathcal{A}ggr_{\mathcal{F}}$\bar{y}.(\phi,\tau)) = max(rk(\phi),rk(τ))+m, where m is the number of first sort variables in y.

Translating L_aggr into L_C

Proposition1: For every formulae \phi(X) of L_aggr(All,All), there exists an equivalent formulae \phi^0(X) of L_C s.t. rk(\phi^0)<= rk(\phi)

L_C is local

- Formulae of L_aggr have finite rank; hence thy are translated into L_C formulae of finite rank.
- It can be shown by a simple induction that those formulae are local.
- The proof is based on Lemma1 – the Permutation Lemma.

The Permutation Lemma

Let D be first-sort, with A=adom(D), and r >0.

If a \isomorphic _{3r+1}^D b , then there exists a permutation

p: A -> A s.t. ac \isomorphic_r^D bp(c) for every c \belongs A.

Putting everything together

- Let e be a pure relational expression of ALG _aggr(ALL,ALL).
- By Theorem 2, it is expressible in L_aggr(ALL,ALL), and by Proposition 1, by a L_C formula of finite rank.
- Hence it is local.

SQL over ordered domains

- So far the only nonnumerical selection was of the form \sigma_{i=j}, testing equality of 2 attributes. We now extend the language to ALG^<_aggr by allowing selections of the form \sigma_{i<j}(e), where both i and j are of type b, and < is some fixed linear ordering on the domain Dom.

Natural Numbers

- Let num = N. We consider a version of ALG_aggr that has the most usual set of arithmetic and aggregate operators: namely, + , < and constants for arithmetic, and the aggregate Σ. This suffices to express aggregates MIN,MAX,COUNT, TOTAL, but certainly not AVG, which, produces rational numbers.

Natural Numbers (Cont.)

- We shall use the notations:

- SQL_N for ALG_aggr ({+,.,<,0,1},{Σ}), and

- SQL_N^< for ALG_aggr^< ({+,.,<,0,1},{Σ}).

- We show how a well-known counting logic FO(C) can be embedded into SQL_N^< .

Definition of FO(C)

- It is a two-sorted logic, with second sort being the sort of natural numbers. That is, a structure D is of the form ({a_1,…,a_n},(1,…,n),<,+,..,1,n,R_1,…,R_t)

Theorem 4.

Over ordered structures,

FO(C) \subset SQL_N^<.

Reachability over ordered domains

- Notice that the reachability query, even over ordered domains of nodes , is order-independent; that is, the result does not depend on a particular ordering on the nodes, just on the graph structure.
- Order-independent queries in SQL_N and SQL_N^< are the same.

Proposition 3

- There exist order-independent non-local queries expressible in SQL_N^<. Thus, there are order-independent SQL_N^< queries not expressible in SQL_N.

Conclusion

- Did SQL3 designers really have to introduce recursion, or is it expressible with what's already there?
- Our results show that they clearly had a good reason for adding a new construct , because:

Conclusion (Cont.)

1. Over unordered types, reachability queries cannot be expressed by the basic SQL *SELECT-FROM-WHERE-GROUPBY-HAVING* statements,

   in fact , all queries expressible by such statements are local.

Conclusion (Cont.)

2. Over ordered domains, with limited arithmetic, reachability queries are most likely inexpressible, but proving this is very hard. Adding more arithmetic operations might help, but only at the expense of encodings which are several thousand digits long – so the new construct is clearly justified.