What is Database Theory?

A collection of studies, often connected to the relational model of data.

- Restricted forms of logic, between SQL and full first-order.
- Dependency theory: generalizing functional dependencies.
- Conjunctive queries (CQ's): useful, decidable special case.
- "Universal relations" fitting a database schema into a single (virtual) relation.

Why Care?

A lot of this work was, quite frankly, done "for the fun of it." However, it turns out to have unexpected applications, as natural ideas often do:

- Information integration:
  - Logic, CQ's, etc., used for expressing how information sources fit together.
  - Recent work using universal-relation too — eliminates requirement that user understand a lot about the integrated schema.
- More powerful query languages.
  - Recursion needed in repositories, other applications.
  - Database logic provided some important ideas used in SQL3 standard: seminaive evaluation, stratified negation.
- Potential application: constraints and triggers are inherently recursive. When do they converge?

Outline of Topics

1. Logic intro, especially logical rules (if-then), dealing with negation.

   - In database logic there is a special semantics frowned upon by Mathematicians, but it works.

2. Logic processing: optimizing collections of rules that constitute a query.

   - "Magic-sets" technique for recursive queries.

3. Conjunctive queries: decidability of containment, special cases.

4. Information-integration architectures: rule expansion vs. systems that piece together solutions to queries from logical definitions of sources.

   - Important CQ application.

5. Universal relation data model: answering queries without knowing the schema.

6. Other stuff if I have time for it and/or there is class interest:

   a) Data mining of databases.

   b) Materialized views, warehouses, data cubes.

Course Requirements

1. The usual stuff: midterm, final, problem sets.

2. A project:

   - Each student should attempt to implement an algorithm for one of the problems discussed in the class.

   - Your choice, but you should pick something that is combinatorially hard, i.e., the problem is dealing efficiently with large cases.

   - I'll suggest some problems as we go, and keep a list on the Web page.

Review of Logic as a Query Language

Datalog programs are collections of *rules*, which are Horn clauses or if-then expressions.

Example

The following rules express what is needed to "make" a file. It assumes these relations or EDB (*extensional database*) predicates are available:

1. $source(F)$: F is a source file, i.e., stored in the file system.
2. $includes(F, G)$: file F includes file G.
3. $create(F, P, G)$: we create file F by applying process P to file G.

```
req(F,F) :- source(F)
req(F,G) :- includes(F,G)
req(F,G) :- create(F,P,G)
req(F,G) :- req(F,H) & req(H,G)
```

Rules

$$\text{Head :- Body}$$

- `:-` is read "if"
- *Atom* = predicate applied to arguments.
- Head is atom.
- Body is logical AND of zero or more atoms.
- Atoms of body are called *subgoals*.
- Head predicate is IDB *intensional database* = predicate defined by rules. Body subgoals may have IDB or EDB predicates.
- Datalog program = collection of rules. One IDB predicate is distinguished and represents result of program.

Meaning of Rules

The head is true for its arguments whenever there exist values for any *local* variables (those that appear in the body, but not the head) that make all the subgoals true.

Extensions

1. Negated subgoals. Example:

```
cycle(F) :- req(F,F) & NOT source(F)
```

2. Constants as arguments. Example:

```
req(F,"stdio.h") :- type(F,"cCode")
```

3. Arithmetic subgoals. Example:

```
composite(A) :- divides(B,A) &
    B > 1 & B <> A
```

   - Opposite of an arithmetic atom is a *relational* atom.

Applying Rules ("Naive Evaluation")

Given an EDB:

1. Start with all IDB relations empty.
2. Instantiate (with constants) variables of all rules in all possible ways. If all subgoals become true, then infer that the head is true.
3. Repeat (2) in "rounds," as long as new IDB facts can be inferred.

- (2) makes sense and is finite, as long as rules are *safe* = each variable that appears anywhere in the rule appears in some nonnegated, nonarithmetic subgoal of the body.
- Limit of (1)–(3) = Least fixed point of the rules and EDB.

Seminaive Evaluation

- More efficient approach to evaluating rules.
- Based on principle that if at round i a fact is inferred for the first time, then we must have used a rule in which one or more subgoals were instantiated to facts that were inferred on round $i - 1$.
-

Thus, for each IDB predicate p, keep both relation P and relation ΔP; the latter represents the new facts for p inferred on the most recent round.

Outline of SNE Algorithm

1. Initialize IDB relations by using only those rules without IDB subgoals.

2. Initialize the Δ-IDB relations to be equal to the corresponding IDB relations.

3. In one round, for each IDB predicate p:

   a) Compute new ΔP by applying each rule for p, but with *one* subgoal treated as a Δ-IDB relation and the others treated as the correct IDB or EDB relation. (Do for *all* possible choices of the Δ-subgoal.)

   b) Remove from new ΔP all facts that are already in P.

   c) $P := P \cup \Delta P$.

4. Repeat (3) until no changes to any IDB relation.

Example

```
(1) req(F,F) :- source(F)
(2) req(F,G) :- includes(F,G)
(3) req(F,G) :- create(F,P,G)
(4) req(F,G) :- req(F,H) & req(H,G)
```

- Assume EDB relations S, I, C and IDB relation R, with obvious correspondence to predicates.
- Initialize: $R \ := \ \Delta R \ := \ \sigma_{\#1=\#2}(S \times S) \cup I \cup \pi_{1,3}(C)$.
- Iterate until $\Delta R = \emptyset$:

  1. $\Delta R \ := \ \pi_{1,3}(R \bowtie \Delta R \cup \Delta R \bowtie R)$
  2. $\Delta R \ := \ \Delta R - R$
  3. $R \ := \ R \cup \Delta R$

Models

Model of rules + EDB facts = set of atoms selected to be true such that

1. An EDB fact is selected true iff it is in the given EDB relation.

2. All rules become true under any instantiation of the variables.

   - Facts not stated true in the model are assumed false.

   - Only way to falsify a rule is to make each subgoal true and the head false.

- *Minimal model* = model + no proper subset is a model.

- For a Datalog program with only nonnegated, relational atoms in the bodies, the *unique* minimal model is what naive or seminaive evaluation produces, i.e., the IDB facts we are *forced* to deduce.

- Moreover, this LFP is reached after a finite number of rounds, if the EDB is finite.

Function Symbols

Terms built from

1. Constants.
2. Variables.
3. Function symbols applied to terms as arguments.

   - Example:

$$addr\big(street(maple), number(101)\big)$$

Example

Binary trees defined by

```
isTree(null)
isTree(node(L,T1,T2)) :-
        label(L) &
        isTree(T1) &
        isTree(T2)
```

If $label(a)$ and $label(b)$ are true, infers facts like

$$isTree\big(node(a, null, null)\big)$$
$$isTree\Big(node\big(b, null, node(a, null, null)\big)\Big)$$

- Application of rules as for Datalog: make all possible instantiations of variables and infer head if all subgoals are true.
- LFP is still unique minimal model, as long as subgoals are relational, nonnegated.
- But LFP may be reached only after an infinite number of rounds.