Datalog: Query Language for Relational Databases

Syntax:

- **Atomic Formula:**
  1. \( p(x_1, \ldots, x_n) \) where \( p \) is a relation name and \( x_1, \ldots, x_n \) are either constants or variables.
  2. \( x \, \text{<op}> \, y \) where \( x \) and \( y \) are either constants or variables and \( \text{<op>} \) is one of the six comparison operators: \(<, \leq, >, \geq, =, \neq\).

Variables that appear only once in a rule can be replaced by anonymous variables (represented by underscores). NOTE: Every anonymous variable is different from all other variables.

- **Datalog rule:**

  \[ p \,:=\, q_1, \ldots, q_n. \]

  where \( p \) is an atomic formula and
  \( q_1, \ldots, q_n \) are either atomic formula or negated atomic formula (i.e. atomic formula preceded by not).
  \( p \) is referred to as the head of the rule.
  \( q_1, \ldots, q_n \) are referred to as subgoals.

- **Safe Datalog rule:** A Datalog rule \( p \,:=\, q_1, \ldots, q_n. \) is safe
  1. if every variable that occurs in a negated subgoal also appears in a positive subgoal and
  2. if every variable that appears in the head of the rule also appears in the body of the rule.

- **Datalog query:**

  set of safe Datalog rules with at least one rule defining the answer predicate, which will correspond to the answers of the query.
Query Examples: (These are the queries from problem 7.18 of the El-Masri/Navathe text).

(1) Get names of all employees in department 5 who work more than 10 hours/week on the ProductX project.
\[
\text{answer}(F,M,L) \leftarrow \text{employee}(F,M,L,S,\_,\_,\_,\_,\_,5), \text{works}
\text{on}(S,P,H), \text{project('ProductX',P,\_,\_)}, H \geq 10.
\]

(2) Get names of all employees who have a dependent with the same first name as themselves.
\[
\text{answer}(F,M,L) \leftarrow \text{employee}(F,M,L,S,\_,\_,\_,\_,\_,\_), \text{dependent}(S,F,\_,\_,\_).
\]

(3) Get the names of all employees who are directly supervised by Franklin Wong.
\[
\text{answer}(F,M,L) \leftarrow \text{employee}(F,M,L,\_,\_,\_,S,\_), \text{employee('Franklin',\_,Wong',S,\_,\_,\_,\_,\_}).
\]

(4) Get the names of all employees who work on every project.
\[
\text{temp1}(S,P) \leftarrow \text{employee}(\_,\_,\_,S,\_,\_,\_,\_,\_,\_,\_), \text{project(\_,P,\_,\_)}. \\
\text{temp2}(S,P) \leftarrow \text{works_on}(S,P,\_). \\
\text{temp3}(S) \leftarrow \text{temp1}(S,P), \text{not temp2}(S,P). \\
\text{answer}(F,M,L) \leftarrow \text{employee}(F,M,L,S,\_,\_,\_,\_,\_,\_), \text{not temp3}(S).
\]

(5) Get the names of employees who do not work on any project.
\[
\text{temp1}(S) \leftarrow \text{works_on}(S,\_). \\
\text{answer}(F,M,L) \leftarrow \text{employee}(F,M,L,S,\_,\_,\_,\_,\_,\_), \text{not temp1}(S).
\]
(6) Get the names and addresses of employees who work for at least one project located in Houston but whose department does not have a location in Houston.

```prolog
temp1(S) :- works_on(S,P,_,), project(_,P,'Houston',_).
temp2(S) :- employee(_,_,S,_,_,_,_,_,_,D),
            not dept_locations(D,'Houston').
answer(F,M,L,A) :- employee(F,M,L,S,_,A,_,_,_,_), temp1(S), temp2(S).
```

(7) Get the names and addresses of employees who work for at least one project located in Houston or whose department does not have a location in Houston. (Note: this is a slight variation of the previous query with 'but' replaced by 'or').

```prolog
temp1(S) :- works_on(S,P,_,), project(_,P,'Houston',_).
temp2(S) :- employee(_,_,S,_,_,_,_,_,_,D),
            not dept_locations(D,'Houston').
answer(F,M,L,A) :- employee(F,M,L,S,_,A,_,_,_,_), temp1(S).
answer(F,M,L,A) :- employee(F,M,L,S,_,A,_,_,_,_), temp2(S).
```

(8) Get the last names of all department managers who have no dependents.

```prolog
temp1(S) :- dependent(S,_,_,_,_).
answer(L) :- employee(_,L,S,_,_,_,_,_,_,_),
            department(_,S,_),
            not temp1(S).
```
create table component (  
  part1  varchar2(20),  
  part2  varchar2(20),  
  amount number(7),  
  attr  char(1)  
);  
create table price (  
  part  varchar2(20),  
  price  number(7)  
);  
insert into component values('engine', 'sparkplug', 4, 'b');  
insert into component values('engine', 'cylinder', 4, 'c');  
insert into component values('engine', 'valve', 4, 'c');  
insert into component values('engine', 'crankshaft', 1, 'c');  
insert into component values('cylinder', 'piston', 1, 'c');  
insert into component values('cylinder', 'connectingrod', 1, 'c');  
insert into component values('valve', 'gasket', 1, 'b');  
insert into component values('valve', 'hanger', 2, 'c');  
insert into component values('crankshaft', 'joint', 8, 'c');  
insert into component values('piston', 'screw', 2, 'b');  
insert into price values ('sparkplug', 10);  
insert into price values ('screw', 2);  
insert into price values ('gasket', 3);  
insert into price values ('bolt', 2);  
sub_part(X, Y, Q, T) :- comp(X, Y, Q, T).  
sub_part(X, Y, Q, T) :- comp(Z, Y, Q2, T), sub_part(X, Z, Q1, T1),  
  Q is Q1 * Q2.  
look_for(P, Y, Q) :- sub_part(P, Y, Q, b).  
basic_comp(P, B, sum(<Q>)) :- look_for(P, B, Q).  
temp_cost(P, X) :- basic_comp(P, B, Q), price(B, C), X is Q * C.  
cost(P, sum(<C>)) :- temp_cost(P, C).