Basis Data Non Relasional
Deductive Database

Tricya Widagdo

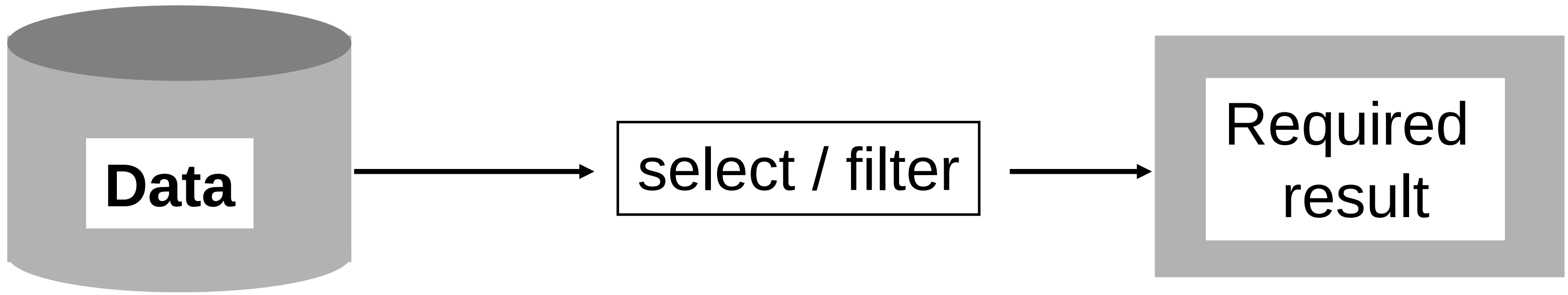
Program Studi Teknik Informatika

Institut Teknologi Bandung

Reference

- Ullman, J. D., "Principles of Database and Knowledge-Base Systems", Vol. I & II, Computer Science Press, 1988
- Elmasri, R., Navathe, S. B., "Fundamentals of Database Systems", 3rd ed., Ch. 25, Addison-Wesley, 2000
- Date, C. J., "An Introduction to Database Systems", 7th ed., Ch. 23, Wiley, 2000
- Silberschatz, A., Korth, H.F., Sudarshan, S., "Database System Concepts", 4th ed., Ch. 5, Mc. Graw Hill, 2002

Relational databases



relations specified ***extensionally***

by evaluating a (relational algebra) formula

Example - database

Person

Name	Dob	Sex	Address
Linda Fox		F	
John Fox		M	
Mary Fox		F	
June Fox		F	
Bill Fox		M	
John Hunt		M	
Jack Hunt		M	
Helen Kent		F	
Dean Kent		F	

Parent

P-name	C-name
Linda Fox	Mary Fox
Linda Fox	June Fox
Linda Fox	Bill Fox
John Fox	Mary Fox
John Fox	June Fox
John Fox	Bill Fox
Mary Fox	John Hunt
Mary Fox	Jack Hunt
June Fox	Helen Kent
June Fox	Dean Kent

Example - query

Query 1: *Find who is Dean Kent's mother*

```
SELECT        Person.Name
FROM          Person, Parent
WHERE         Person.Sex = 'F' AND
              Person.Name = Parent.Name
AND
              Parent.Child = 'Dean Kent' ;
```

Query 2: *Find who is Dean Kent's grandmother, on his mother's side*

```
SELECT        Person.Name
FROM          Person, Parent
WHERE         Person.Sex = 'F' AND
              Person.Name = Parent.Name AND
              Parent.Child IN
              (SELECT     Person.Name
               FROM       Person, Parent
               WHERE      Person.Sex = 'F' AND
                          Person.Name = Parent.Name AND
                          Parent.Child = 'Dean Kent');
```

Example - query (2)

Query 3: *Find all mothers (and their addresses) who have at least one daughter*

```
SELECT        Name, Address
FROM          Person
WHERE         Sex = 'F'  AND  EXISTS
              (SELECT     *
               FROM       Parent
               WHERE      Person.Name = Parent.Name AND
                          Parent.Child IN
                          (SELECT     *
                           FROM       Person
                           WHERE      Sex = 'F')) ;
```

Query 4: *Find all grandmothers (and their addresses) who have at least one grand daughter*

homework

Query 1 - conclusion

- the answer to 'grandmother' queries would have been easier if a 'Grandparent' relation had existed
  - ***improper*** solution if the relation would have been defined ***extensionally*** - redundancies (exemplified in the following slides)
  - therefore ***intensional definitions are required*** (exemplified on the following slides)
- could the query language be simpler?
  - yes; Datalog (Prolog like), for instance

‘Grandparent’ relation - extensionally

Parent

Name	Child
Linda Fox	Mary Fox
Linda Fox	June Fox
Linda Fox	Bill Fox
John Fox	Mary Fox
John Fox	June Fox
John Fox	Bill Fox
Mary Fox	John Hunt
Mary Fox	Jack Hunt
June Fox	Helen Kent
June Fox	Dean Kent

Grandparent

Name	Grandchild
Linda Fox	John Hunt
Linda Fox	Jack Hunt
Linda Fox	Hellen Kent
Linda Fox	Dean Kent
John Fox	John Hunt
John Fox	Jack Hunt
John Fox	Helen Kent
John Fox	Dean Kent

not really a good solution

‘Grandparent’ relation - intensionally

Parent

Name	Child
Linda Fox	Mary Fox
Linda Fox	June Fox
Linda Fox	Bill Fox
John Fox	Mary Fox
John Fox	June Fox
John Fox	Bill Fox
Mary Fox	John Hunt
Mary Fox	Jack Hunt
June Fox	Helen Kent
June Fox	Dean Kent

A is **Grandparent** of B
IFF

/* there is a C such that */
A is the Parent of C
AND
C is the Parent of B

Activity !!

Reconsider the issues discussed so far on the Person' relation below, which substitutes the previous (page 4) Person and Parent relations.

Person'

Name	Dob	Sex	Address	Mother	Father
Linda Fox		F		null	null
John Fox		M		null	null
Mary Fox		F		Linda Fox	John Fox
June Fox		F		Linda Fox	John Fox
Bill Fox		M		Linda Fox	John Fox
John Hunt		M		Mary Fox	null
Jack Hunt		M		Mary Fox	nul
Helen Kent		F		June Fox	null
Dean Kent		F		June Fox	null

Example – database #2

Person

Name	Dob	Sex	Address
Linda Fox		F	
John Fox		M	
Mary Fox		F	
June Dyer		F	
Bill Dyer		M	
John Hunt		M	
Rose Hunt		M	
Helen Kent		F	
Dean Kent		F	

Parent

Name	Child
Linda Fox	Mary Fox
John Fox	Mary Fox
Mary Fox	John Hunt
Mary Fox	Rose Hunt
John Hunt	June Dyer
June Dyer	Bill Dyer
Jack Hunt	Helen Kent
Jack Hunt	Dean Kent

Example – query (3)

Query 5: *Find who Bill Dyer's ancestors are*

- Recursive query
  - Not possible to answer in the relational model
  - Another formalism is necessary – which allows recursive definitions
- Recursive definition

A is ***ancestor*** of P IFF
A is *parent* of P

A is ***ancestor*** of P IFF
∃ B (B is *ancestor* of P AND A is *parent* of B)

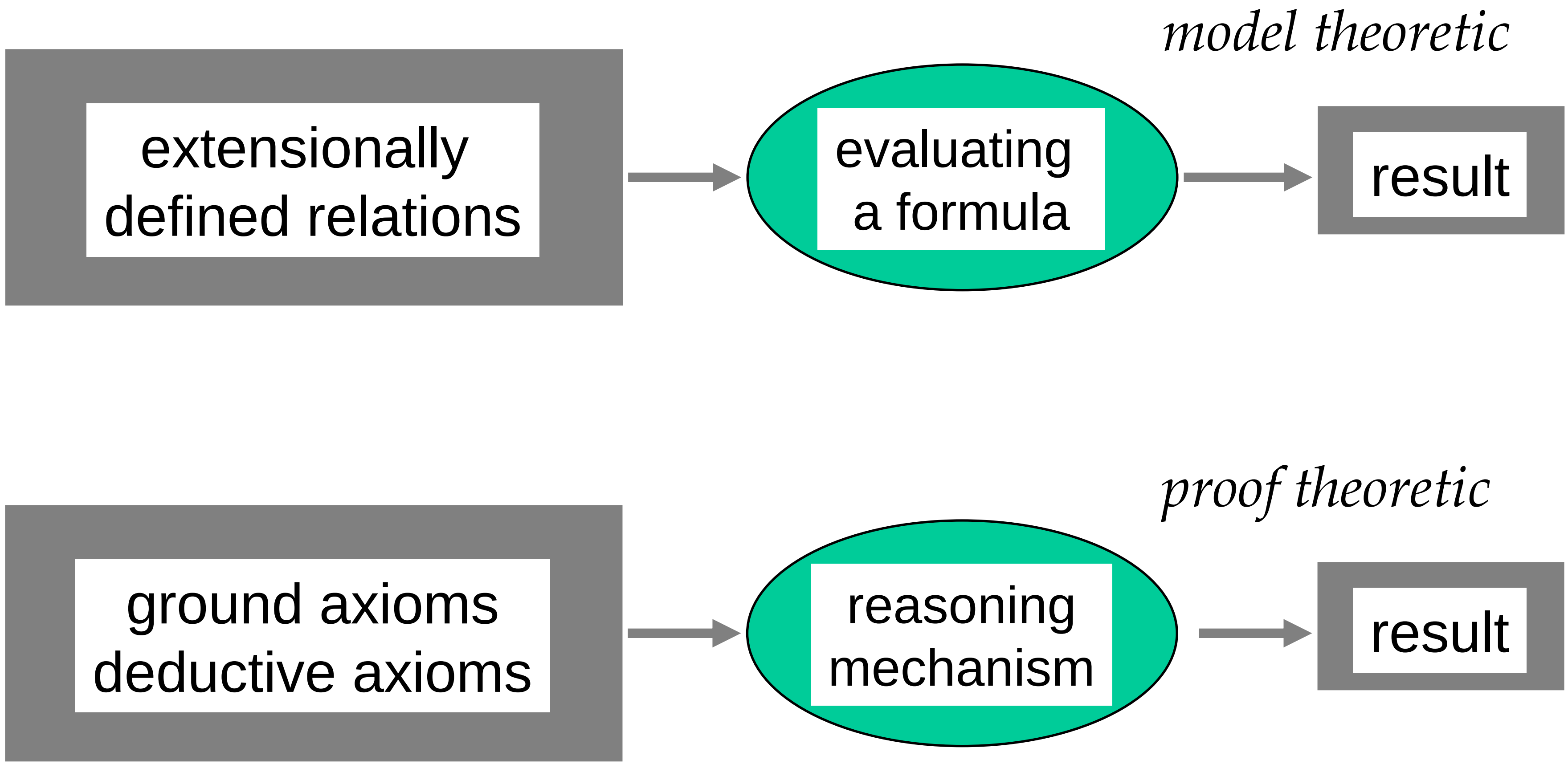
Intermediate conclusion

- a mechanism is required to
  - allow relations to be intensionally defined
  - allow new facts to be deduced from the database
    - based on the defined facts and rules
    - by means of some reasoning methods
- *note*
    - *the motivation for deductive databases is more complex (i.e. not reduced to the above two aspects)*

Deductive database - informal definition

- a database that supports the definition of both facts (extensionally defined relations) and rules (intensionally defined relations) and which supplies a reasoning mechanism by means of which existing or new facts can be deduced from the database
- a deductive database uses logic (first order predicate logic) as its representational formalism

Relational vs Deductive databases



Deductive Databases

- An area that is at the intersection of databases, logic, and artificial intelligence (or knowledge-bases)
- Emphasizing in mechanism to get new knowledge from stored data by adding 'relationships' between data (by means of rules)
- A deductive database system is a database system which includes capabilities to define (deductive) rules which can deduce or infer additional information from the facts stored in the database
  - Rules are specified through a declarative language
  - An inference engine, which is included in the system, can deduce new facts from the database by interpreting existing rules
- Deductive database systems (also called logic databases) are not the same with knowledge-base systems even though both incorporate reasoning and inferencing capabilities

Deductive Databases (2)

- Deductive databases use two main types of specifications: facts and rules
  - Facts are specified in a manner similar to the way relations are specified
    - But, it is not necessary to include the attribute names, therefore position of values in a fact is important
    - Extensional database
  - Rules are used to define *virtual relations* which can be formed from the facts by applying inference mechanism based on the rule specifications
    - The main difference between rules and views is that rules may involve recursion and hence may yield virtual relations that cannot be defined in terms of standard relational views
    - Intensional database

Prolog/Datalog

- The model used in deductive databases is related with logic programming and Prolog
- Datalog is a variation of Prolog which has been devoted to handling large volumes of data stored in a relational database
- The notation used in Prolog/Datalog is based on providing predicates with unique names
- A predicate has an implicit meaning, which is suggested by the predicate name, and a fixed number of arguments
- The predicate's type is determined by its arguments:
  - If the arguments are all constant values, the predicate simply states that a certain fact is true
  - If the predicate has variables as arguments, it is either considered as a query or as part of a rule or constraint

Prolog/Datalog (2)

- Conventions:
  - Predicate symbols, function symbols, and constants begin with a lower-case letter (exception: constants are also permitted to be integers)
  - Variables must begin with a capital letter

Notation:

Facts:

```
predicate_name(const1,…,constn)
```

Rules:

```
head :- body
```

- :- is read as "if"
- *left hand side* (LHS) is the conclusion, which will be true if all the RHS predicates are true
- The commas between the RHS predicates may be read as meaning "and"

Prolog/Datalog (3)

- If we have two (or more) rules with the same LHS, it is equivalent to applying logical “or” operator to the rules
- A recursive rule is a rule where one of the rule budy predicates in the RHS is the same as the rule head predicate in the LHS (simple definition)

Queries:

- If one or more arguments are variables, the query will return the values for the variables
- If all the arguments are constants, the query will return *true* or *false*

- Example:

  EMPLOYEES(NAME, DEPT, SALARY, ADDRESS)

  – SQL syntax to create a view that only contains NAME, DEPT, and ADDRESS:

    CREATE VIEW SAFE-EMPS BY SELECT NAME, DEPT, ADDRESS
    FROM EMPLOYEES;

  – Equivalent logical statement:

    safe-emps(N,D,A) :- employees(N,D,S,A)

Prolog/Datalog (4)

Example:

Facts:

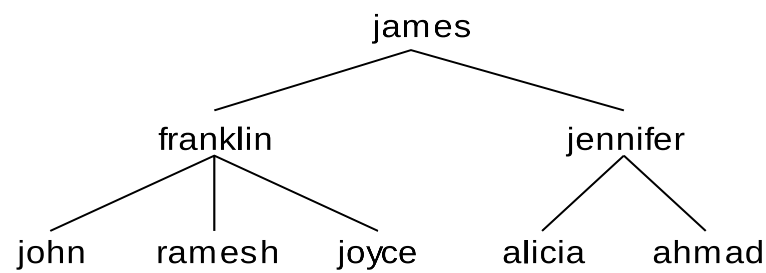
```
supervise(franklin,john).
supervise(franklin,ramesh).
supervise(franklin,joyce).
...
```

Rules:

```
superior(X,Y) :-
  supervise(X,Y).
superior(X,Y) :-
supervise(X,Z),superior(Z,Y).
subordinate(X,Y) :-
  superior(Y,X).
```

Queries:

```
superior(james,Y)?
superior(james,joyce)?
```



The meaning of logical rules

There are three alternative ways to define the "meaning" of rules

- *Proof-theoretic* interpretation
  - From the facts in the database, we see what other facts can be proved using the rules in all possible ways
  - Using the rules in the "forward" direction only, by inferring left sides (consequents, or conclusions) from right sides (antecedents or hypotheses)
- *Model-theoretic* interpretation
  - We see rules as defining possible worlds or "models"
  - An interpretation of a collection of predicate assigns truth or falsehood to every possible instance of those predicates, where the predicates' arguments are chosen from some infinite domain of constants
    - Usually, an interpretation is represented by its set of true instances
  - A model is said to be a minimal model consistent with a database state if we cannot make any true fact false and still have a model

The meaning of logical rules (2)

- *Computational Definitions of Meaning*
  - Providing an algorithm for "executing" logical rules to tell whether a potential fact (predicate with constants for its arguments) is true or false
    - E.g. translating rules into sequences of operations in relational algebra → the result is a minimal model
- Comparison of "Meanings"
  - Which is the "best" meaning for a logic program?

The Datalog data model

- Datalog is a version of Prolog suitable for database systems
- It differs from Prolog in:
  - Datalog allows only variables and constants as arguments (function symbols are not allowed)
  - The 'meaning' of Datalog programs follows the model-theoretic point of view, while Prolog has a computational 'meaning'
- The underlying mathematical model of data is essentially that of relational model
  - Relations in the set-of-list sense
    - Components appear in a fixed order
    - Reference to a column is only by its position among the arguments

Extensional and intentional predicates

- There are two ways relations can be defined:
  - Extensional database (EDB) relations: predicates whose relations are stored in the database
  - Intentional database (IDB) relations: predicates whose relations are defined by logical rules
- Assumption: each predicate symbol either denotes an EDB relation or an IDB relation, but not both

Atomic formulas

- Datalog programs are built from atomic formulas:
  - An atomic formula is a predicate symbol with a list of arguments, e.g. $p(A_1, \ldots, A_n)$, where *p* is the predicate symbol
    - Each predicate symbol is associated with a particular number of arguments that it takes: $p^{(k)}$ denotes a predicate with *k* arguments
  - An atomic formula denotes a relation; it is the relation of its predicate restricted by:
    - Selecting for equality between a constant and the component or components in which that constant appears, and
    - Selecting for equality between components that have the same variable
  - Atomic formulas can also be built using built-in predicates, i.e. comparison predicates.

Clauses and Horn clauses

- A literal is either a positive literal (atomic formula) or a negative literal (negated atomic formula)
- A clause is a sum (logical OR) of literals
- Formulas are converted into clausal form before they can be expressed in Datalog
  - Only formulas given in a restricted clausal form, called Horn clauses, can be used in Datalog
- Characteristics of formulas in clausal form:
  - All variables are universally quantified
    - The quantifiers can be removed from the formula
  - A formula is made up of a number of clauses
  - The clausal form of a formula is a conjunction of clauses

Clauses and Horn clauses (2)

- A Horn clause is a clause with at most one positive literal
  - A single positive literal
    - Considered as a fact, e.g. *p(X,Y)*
  - One or more negative literals with no positive literal
    - Considered as integrity constraint
  - A positive literal and one or more negative literals
    - Considered as rule
    - Example: the following horn clause

$$\overline{p}_1 \vee \ldots \vee \overline{p}_n \vee q$$

can be translated to the following rule

$$q \text{ :- } p_1 , \ldots , p_n .$$

q is called the head of the rule, and each p_i is called subgoal

A collection of Horn clause is termed a logic program

Dependency graphs and recursion

- A dependency graphs is used to show the way predicates in a logic program depend on one another
  - Ordinary (non built-in) Predicates are represented using nodes
  - An arc is formed from predicate *p* to predicate *q* if there is a rule with a subgoal whose predicate is *p* and a head whose predicate is *q*
- A logic program is *recursive* if its dependency graph has one or more cycles
  - All the predicates that are on one or more cycles are said to be *recursive predicates*

Safe rules

- A rule is said to be safe if it operates on finite relations
- One simple approach to avoiding rules that create infinite relations from finite ones is to insist that each variable appearing in the rule be limited
- Limited variables:
  - Appears as an argument in an ordinary predicate of the body
  - Any variable X that appears in a subgoal $X=a$ or $a=X$, where a is a constant
  - Variable X that appears in a subgoal $X=Y$ or $Y=X$ where Y is a variable already known to be limited
- A rule is safe if all its variables are limited

Basic Inference Mechanisms

- Bottom-up inference mechanisms (Forward Chaining)
  - Starts with the facts and applies the rules to generate new facts
    - New facts are checked against the query predicate goal for a match
  - A search strategy to generate only the facts that are relevant should be used $\rightarrow$ efficiency
- Top-down inference mechanisms (Backward Chaining)
  - Starts with the query predicate goal and attempts to find matches to the variables that lead to valid facts
    - Binding process
  - Strategies for compound goals:
    - Depth-first search: search for matches are done consecutively
      - The order of subgoals for recursive rules is important
    - Breadth-first search: search for matches proceed in parallel

Nonrecursive rules

- Nonrecursive rules can be ordered:
  - If there is an arc from p_i to p_j in the dependency graph, then $p_i < p_j$.
  - Computation of relations is done in that order
- The computation of the relation for p_i is divided into two steps:
  - For each rule *r* with p_i at the head, compute the relation corresponding to the body of the rule
    - Compute the natural join of relations corresponding to its subgoals
    - This relation has one component for each variable of *r*
  - Relation for p_i can be computed by:
    - Projecting the relation for each p_i's rules onto the components corresponding to the variables of the head
    - Taking the union over all rules with p_i in the head

Relation defined by a rule body

- The relation for a rule *r* is defined to have the scheme $X_1, \ldots, X_m$, where the X_i's are the variables of the body of *r*, in some selected order
- Suppose that $p_1,\ldots,p_n$ is the list of all predicates appearing in the rule body, and $P_1,\ldots,P_n$ are relations related to the predicates. A subgoal *S* of rule *r* is made true by a substitution if the following hold:
  - If *S* is ordinary, $p(b_1,\ldots,b_k)$, then $(b_1,\ldots,b_k)$ should be a tuple in the relation *P* corresponding to *p*.
  - If *S* is built-in, $b\theta c$, then the arithmetic relation $b\theta c$ should be true.

Constructing A Relational Algebra Expr.

- INPUT: the body of a datalog rule r, which consists of subgoals $S_1,\dots,S_n$ involving variables $X_1,\dots,X_m$. For each subgoal, there is a relation already computed.
- OUTPUT: An expression of relational algebra, which is called EVAL-RULE($r,R_1,\dots,R_n$) that computes from relations $R_1,\dots,R_n$ a relation $R(X_1,\dots,X_m)$.
- METHOD:
  - For each ordinary S_i, let Q_i be the expression $\pi_{Vi}(\sigma_{Fi}(R_i))$, where:
    - V_i is a set of distinct variables
    - F_i is the conjunction (logical AND) of the following conditions:
      - If position k has a constant a, then F_i has the term $\$k=a$.
      - If position k and l both contain the same variable, then F_i has the term $\$k=\$l$.

Constructing A Relational Algebra Expr. (2)

- METHOD: (cont.)
  - For each variable X not found among the ordinary subgoals, compute an expressing D_X that produces a unary relation containing all the values that X could possibly have.
    - Since r is safe, there is a variable Y to which X is equated:
      - If $Y=a$ is a subgoal, then let D_X be the constant expression $\{a\}$.
      - If Y appears as the *jth* argument of ordinary subgoal S_i, let D_X be $\pi_j(R_i)$.
  - Let E be the natural join of all the Q_i's and D_X's.
  - Let EVAL-RULE($r, R_1, \ldots, R_n$) be $\sigma_F(E)$, where F is the conjuction of $X\theta Y$ for each built-in subgoals and E is the expression that has been constructed.

Rectified Rules

- The rules for predicate *p* are rectified when all their heads are:
  - Identical
  - No constants arguments
  - All variables are distinct
- How to rectify rules?
  - Introduce new variables for each of the arguments of the head predicate
  - Introduce built-in subgoals into the body to enforce whatever constraints the head predicate formerly enforced through constants and repetitions of variables

Computing Relations for Nonrecursive Predicates

- INPUT: a non recursive datalog program and a relation for each EDB predicate appearing in the program.
- OUTPUT: for each IDB predicate *p*, an expression of relational algebra that gives the ralation for *p* in terms of the relations $R_1,\ldots,R_m$ for the EDB predicates.
- METHOD:
  - Rectify all the rules
  - Define the order of predicates (may use dependency graph)
  - Form expression for relation P_i (for p_i) as follows:
    - If p_i is an EDB predicate, let P_i be the given relation for p_i.
    - If p_i is an IDB predicate then:
      - For each rule *r* having p_i as its head, compute the R.A. expression
      - Substitute all the appearances of IDB relations with their R.A. expressions
      - Take the expression for P_i to be the union over all rules *r* for p_i (after performing projection, and if necessary, renaming of variables)

Recursive query processing terminologies

- Two approaches:
  - Pure evaluation approach: creating a query evaluation plan that produces an answer
  - Rule rewriting approach: optimizing the plan into a more efficient strategy
- A rule is said to be *linearly recursive* if the recursive predicate appears once and only once in the RHS
  - *left linierly recursive*:

    `ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y)`
  - *right linierly recursive*:

    `ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)`

Terminologies…

- Given a Datalog program, the :- symbol can be replaced by an equality to form Datalog equation, without any loss of meaning
  - In a set of relations for the EDB predicates, say $R_1,\ldots,R_n$, a fixed point of the Datalog equation is a solution for the relations corresponding to the IDB predicates
  - The fixed point along with the EDB relations forms a model of the Datalog program (but not vice versa)
  - Each Datalog program has a unique minimal model containing any given EDB relations, and this also corresponds to the unique minimal fixed point.

Recursive rules

- The proof-theoretic approach can be used to derive some facts using a rule, and use newly derived facts in the body to derive more facts
- Given EDB relations $R_1,\ldots, R_n$ with IDB relations $P_1,\ldots,P_m$ to be computed. The set of provable facts for each predicate p_i can be expressed as:

  $$P_i = \text{EVAL}(p_i, R_1,\ldots,R_n, P_1,\ldots,P_m)$$

  where EVAL is the union of EVAL-RULE for each of the rules for p_i, projected onto the variables of the head.
- Starting from all P_i's empty, execute the assignment for each i until no more facts can be added to any of the P_i's

Naïve strategy

- Pure evaluation, bottom up strategy which computes the least model of a Datalog program
- An iterative strategy
  - At each iteration all ruleas are applied to the set of tuples produced thus far to generate all implicit tuples
- Algorithm:
  - INPUT: A collection of datalog rules with EDB predicates *r1,…,rk* and IDB predicates *p1,…,pm*. Also a list of relations *R1,…,Rk*
  - OUTPUT: The least fixed point solution
  - METHOD:
    - Set up equations for the rules
    - Perform the following algorithm

$$
\begin{array}{l}
\texttt{for } i = 1 \texttt{ to } m \texttt{ do } P_i = \varnothing; \\
\texttt{repeat} \\
\quad \texttt{for } i = 1 \texttt{ to } m \texttt{ do } Q_i = P_i; \\
\quad \texttt{for } i = 1 \texttt{ to } m \texttt{ do } P_i = \mathrm{EVAL}(p_i, R_1, \ldots, R_k, Q_1, \ldots, Q_m); \\
\texttt{until } P_i = Q_i \texttt{ for all } i,\ 1 \le i \le m
\end{array}
$$

Seminaive strategy

- Designed to eliminate redundancy in the evaluation of tuples at different iterations
- Algorithm:
  - INPUT: A collection of datalog rules with EDB predicates *r1,…,rk* and IDB predicates *p1,…,pm*. Also a list of relations *R1,…,Rk*
  - OUTPUT: The least fixed point solution
  - METHOD:

```
for i = 1 to m do
      ΔP_i = EVAL(p_i,R_1,…,R_k,∅,…,∅);
      P_i = ΔP_i;
repeat
      for i = 1 to m do ΔQ_i = ΔP_i;
      for i = 1 to m do
       ΔP_i = EVAL-INCR(p_i,R_1,…,R_k,P_1,…,P_m,ΔP_1,…,ΔP_m) - P_i;
      for i = 1 to m do P_i = P_i ∪ ΔP_i;
until ΔP_i = ∅ for all i, 1≤i≤n
```

Magic set rule rewriting technique

- Problem: frequently, a query asks not for the entire relation corresponding to an intentional predicate, but for a small subset of this relation.
- E.g.:

```
sg(X,Y) :- flat(X,Y).
sg(X,Y) :- up(X,U), sg(U,V), down(V,Y).
```

  For query like `sg(john,Z)`, the query must examine part of the database that is *relevant*.
- Solution: combining top-down and bottom-up approach.
  - Rewrite the rules as a function of the query form only
- A typical magic sets transformation of the previous rules would be:

```
sg(X,Y) :- magic-sg(X), flat(X,Y).
sg(X,Y) :- magic-sg(X), up(X,U), sg(U,V), down(V,Y).
magic-sg(U) :- magic-sg(X), up(X,U).
magic-sg(john).
```

Negation

- A deductive database query language can be enhanced by permitting negated literals in the bodies of rules in programs
  - Rules with negated subgoals are not Horn clause
  - In general, the intuitive meaning of a rule with one or more negated subgoals is that we should complement the relations for the negated subgoals
    - Complement of a relation is not a well defined term
- In the presence of negated literals, a program may not have a minimal or least model
- To deal with the problem of many minimal fixed points, what is permitted is only stratified negations

Stratified Negation

- Rules are *stratified* if whenever there is a rule with head predicate *p* and a negated subgoal *q*, there is no path in the dependency graph from *p* to *q*
- E.g.:

```
r1: ancestor(X,Y) :- parent(X,Y).
r2: ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
r3: nocyc(X,Y) :- ancestor(X,Y), ¬(ancestor(Y,X)).
```

- A bottom up evaluation of the rules would first compute a fixed point of non-negated rules before computing negated rules
  - E.g., for the above rules, r3 is applied only when all the ancestor facts are known

Finding Stratification

- In a logic program with negative subgoals, the predicates can be grouped into *strata*
  - If a predicate *p* has a rule with a subgoal that is a negated *q*, then *q* is in a lower stratum than *p*
  - If predicate *p* ha a rule with a subgoal that is a nonnegated *q*, then the stratum of *p* is at least as high as the stratum of *q*
- The strata give an order in which the relations for the IDB predicates may be computed
  - By following it, we may treat any negated subgoals as if they were EDB relations

Finding Stratification 2

- Algorithm:
  - INPUT: A set of datalog rules, possibly with negative subgoals
  - OUTPUT: Are the rules stratified? If so, also produce stratification
  - METHOD:

```
for each predicate p do stratum[p] := 1;
repeat
  for each rule r with head predicate p do begin
    for each negated subgoal or r with predicate q do
      stratum[p] := max(stratum[p], 1+stratum[q]);
     for each nonnegated subgoal or r with predicate q do
      stratum[p] := max(stratum[p], stratum[q])
  end
until there are no changes to any stratum or
      some stratum exceeds the number of predicates
```