EXAMPLE: Given relations

employee(Emp, Salary, DeptNo)
and dept(DeptNo, Mgr)

find all (employee, manager) pairs:

SQL: SELECT Emp, Mgr
FROM employee, dept
WHERE employee.DeptNo = dept.DeptNo

RA: Emp, Mgr
(employee X dept)
WHERE employee.DeptNo = dept.DeptNo
FROM employee, dept
SELECT Emp, Mgr

Relational Algebra (RA) •
Relational Calculus (RC) •
Datalog •

Datalog: boss(Emp, Mgr) -> employee(Emp, Salary, DeptNo)

Example:

\begin{align*}
\forall x, y, z \in \text{employees} & \\
& x.\text{Emp} = y.\text{Emp} \\
& y.\text{Manager} = z.\text{Emp} \\
& \text{SELECT} \ldots \text{WHERE} \ldots
\end{align*}

Query Languages for Relational Databases •
A relational database is given as a set of facts:

- Employee(john, toy's, 5000, cs)
- Employee(john, toy's, 40000, toy's)
- Employee(mary, 6500, cs)
- …
- Department(cs, mary)
- …

A Datalog program defines views by means of rules of the form Head \rightarrow Body:

- \text{boss}(\text{Emp}, \text{Mgr}) \rightarrow \text{employee}(\text{Emp}, \text{Salary}, \text{DeptNo}) \rightarrow \text{dept}(\text{DeptNo}, \text{Mgr})
- \text{highpaid}(\text{Emp}) \rightarrow \text{employee}(\text{Emp}, \text{Salary}) \rightarrow \text{Salary} > 6000
- \text{employee}(\text{Emp}, \text{Salary}, \text{DeptNo}) \rightarrow \text{dept}(\text{DeptNo}, \text{Mgr})
- \text{answer}(\text{Emp}, \text{Mgr}) \rightarrow \text{employee}(\text{Emp}, \text{Salary}, \text{DeptNo}) \rightarrow \text{dept}(\text{DeptNo}, \text{Mgr})

A query is a view with a distinguished answer relation:

- \text{answer}((\text{Emp}, \text{Mgr}), \text{employee}((\text{Emp}, \text{Salary}, \text{DeptNo}), \text{dept}(\text{DeptNo}, \text{Mgr}))

A Datalog database contains defined relations (facts) and defined views (rules). The database contains:

- Employee(john, toy's, 5000, cs)
- Employee(john, toy's, 40000, toy's)
- Employee(mary, 6500, cs)
- …

Notation:

- Lowercase: relation names (employee, dept), aka data values (john, toys)
- UPPERCASE: Capitalized: variables (Emp, X), don't care (\_\_\_)

DATALOG SYNTAX
Relational operations have concise representations:

- \( \text{select}(X) \): \( \text{SELECT } \) some tuples from \( X \)
- \( \text{project}(X) \): \( \text{PROJECT } \) the first argument
- \( \text{join}(X,Y,Z) \): \( \text{JOIN } \) \( X \), \( Y \), \( Z \)
- \( \text{product}(X,Y) \): \( \text{PRODUCT } \) of \( X \) and \( Y \)
- \( \text{intersect}(X) \): \( \text{INTERSECTION } \) of \( X \)
- \( \text{difference}(X) \): \( \text{DIFFERENCE } \) of \( X \)
- \( \text{union}(X) \): \( \text{UNION } \) of \( X \)
- \( \text{set}(X,Y) \): \( \text{SET } \) \( X \) \( \text{from } \) \( Y \)

Rules have a "logical reading" (i.e., rules are formulas):

\[
\cdot (X) b \land (X)d \rightarrow (X) \text{union } X \wedge \\
\cdot (X)b \lor (X)d \rightarrow (X) \text{diff } X
\]

\[
(X)b \land \cdots \% \\
\cdots (X)d \text{ union } \%
\]

\[
(X)b \set \text{DIFFERENCE } \%
\]

\[
(X)b \lor (X)d \text{ INTERSECTION } \%
\]

\[
(X)b \land (X)d \text{ PRODUCT } \%
\]

\[
(X)b \land (X)d \text{ JOIN } \text{ on the first argument } \%
\]

\[
(X)b \land (X)d \text{ SELECT } \text{ some tuples from } \%
\]

Relational operations have concise representations! Examples:
Relations can be defined (directly or indirectly) in terms of themselves:

\[(\text{head} \rightarrow \text{body}) \circ \text{head} \subseteq (\text{head} \rightarrow \text{body}) \circ \text{head}\]

The sequence \(I^0 \subseteq I^1 \subseteq \cdots \) converges to the least fixpoint of the \(d_L\) operator:

\[
(uI)^{d_L} \cap uI =: 1^+uI
\]

\[
\emptyset =: 0I
\]

Bottom-up Evaluation (Fixpoint Semantics):

- Apply rules iteratively (in so-called \(L\)-rounds) until a fixpoint is reached.
- Immediate Consequences operator:

\[
d_L \bigcup (d_L)^{1}\bigcup (d_L)^{2} \cdots
\]

Prov: Positive
Exercise: how about the following rule?

\[
\text{tc}(X, Y) \rightarrow \text{tc}(X, Z) \land \text{tc}(Z, Y).
\]

If the longest path in \( e/2 \) has length \( n \), then \( O(n) \) rounds are needed!

\[
\{(p, q) : p \in e, q \in e^\star \} \cap \{(p, q) : p \in e, q \in e^\star \} = \emptyset
\]
A model $W$ of $P$ is **minimal** if there is no other model $W'$ $\subseteq W$.

A model $W$ of a program $P$ is **minimal** if there is no other model $W'$ $\subseteq W$.

Given the rules of $P$, minimal models:

- From the possible different models find the "intuitive", "intended" models, i.e., which make true only what is "strictly necessary".
- ...models, i.e., which make true only what is "strictly necessary".

Then, an interpretation can be written as a set of true tuples:

(Here, we're dealing with Herbrand interpretations and models, i.e., which interpret syntactically constants.)

All other tuples are regarded as false.

A model of a program $P$ and its rules is an interpretation of the relations which satisfies all rules.

Rules can be seen as first-order formulas:

\[(\forall X : X)(\exists X) X \quad \rightarrow \quad (X)(\exists X) X \quad : \quad (X)\]
Theorem: If $P$ is positive (does not contain negation), then there is a unique minimal model $\{q, a\} = dM$. •

For positive programs, the intersection of two models of $P$ is again a model of $P$. $dM \subseteq dM$. •

$P$: $b$ :- $a$; $b$. $c$ :- $d$. $\models I = \{b\}$ (because $a$ should be true)

$M := \{a, b, c, d\}$ is a model, but not minimal (because $a$ is false, everything else is true). $M$ is the intersection of all Herbrand models of $P$.

THEOREM: The minimal model coincides with the least fixpoint of $L$. $dL \models f = dM$. •

$\{p, c\}$ is a model, but not minimal, since $\{p, c\} = NW \neq I$. •

EXAMPLE: $P$: $b$ :- $a$; $b$. $c$ :- $b$. $d$. $\models I = \{a, b\}$ is a model, but not minimal, since $\{c, d\}$ are not supported by any rule.
Programs with negation may have several minimal models:

\[ \text{not} \text{tc}(X,Y) \]

... % complement of tc/2
...

... % ONLY NOW can we compute the

node \( (X), \text{node}(Y) \),

\[ \text{non-tc}(X',Y) \]

... %

... % recursively

\[ \text{tc}(X,Y) \]

... %

\[ \text{tc}(Z,Y) \]

... %

\[ \text{tc}(X,Z) \]

... %

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]

\[ \text{node} \]
There is much more to say about:

1. Negation: for non-stratified programs (i.e., with negative cyclic dependencies) there exist several semantics with different advantages/disadvantages:

   - well-founded semantics (skeptical declarative), stable semantics (credulous declarative),

2. Translation of RA/RC to Datalog (e.g., universal quantification is expressed using negated existential quantification and auxilliary predicates)

3. Power of using function symbols (encoding proof tree lineage, object fusion, grouping/aggregation)

4. For (1) and (2) see textbook [Foundations of Databases, Abiteboul, Hull, Vianu, Addison-Wesley]

   ...