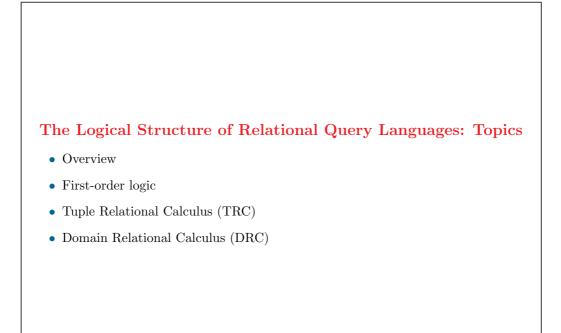
# Course Notes on

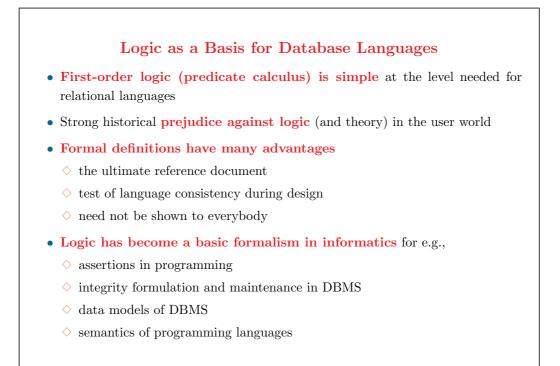
# The Logical Structure of Relational Query Languages



# On the Way to SQL: Relational Calculi

- Historically, SQL was a major advance over older database languages (like DL/I of IMS or DDL, DML of CODASYL DBTG) because SQL is far easier to use
- To effectively master and use SQL up to relational completeness, first mastering first-order logic makes things significantly easier





# **Relational Calculi**

- More used than the algebra as a basis for user languages
- Directly based on first-order logic  $\Rightarrow$  regular, systematic structure
- Less **procedural** than the algebra : **what** versus **how**
- Relational completeness:
  - $\diamond\,$  DRC, TRC, and algebra have same expressive power
  - $\diamond$  SQL is slightly more powerful: some computation, ordering, etc.

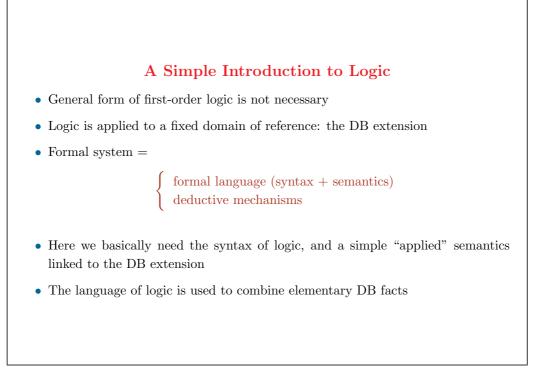
#### 4

## TRC and DRC

- Domain Relational Calculus (DRC)
  - ♦ Most similar to logic as a modeling language
  - ♦ Typical modeling formalism in AI and natural-language studies: data is viewed as objects with properties

#### • Tuple Relational Calculus (TRC)

- ◇ Reflects traditional pre-relational file structures
- $\diamond~$  Closer to a view of relations implemented as files



- $\mathbf{6}$
- Simple and intuitive introductions to logic:
  - ◊ Introduction to Logic for Liberal Arts and Business Majors, by S. Waner and R. Costenoble, http://www.hofstra.edu/ matscw/logicintro.html, July 1996.
  - ♦ Sweet Reason: A Field Guide to Modern Logic, by T. Tymoczko and J. Henle, Springer Textbooks in Mathematical Sciences, ISBN 0-287-98930-7, Springer, 2nd ed., 1999

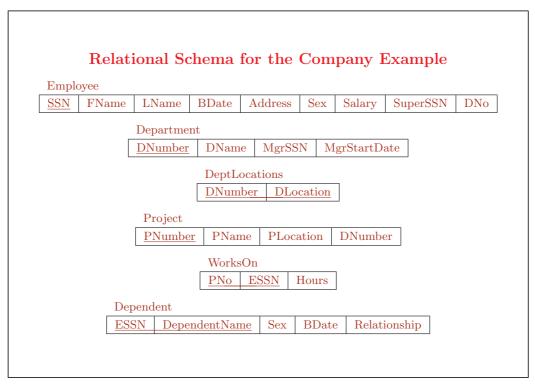
# The Structure of First-Order Logic

- The universe of reference is the current database
- Elementary propositions: express assertions that are true or false in the universe
- **Propositional connectives**  $(\land, \lor, \rightarrow, \neg, \leftrightarrow)$  combine propositions

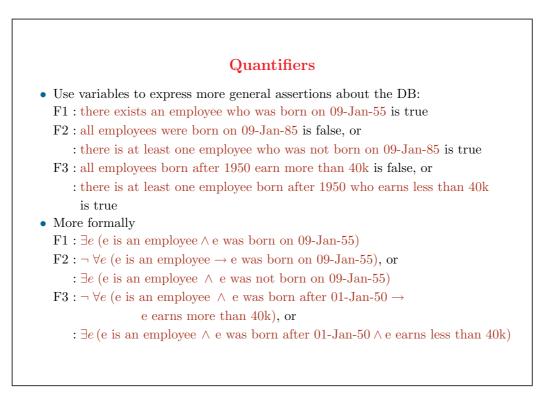
P				$P \to Q$			
T	T	T	T	T	F	Т	
T	F	F	T	F	F	F	
F	T	F	T	T	T	F	
F	F	F	F	T F T T	T	T	

 $\overline{7}$ 

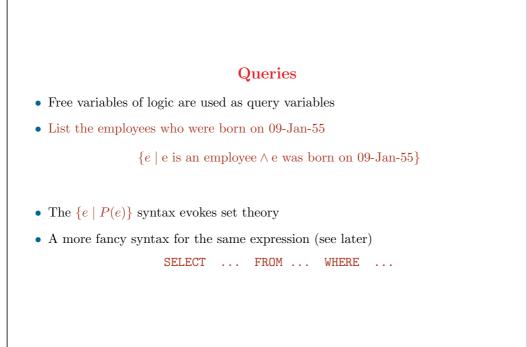
- Elementary propositions:
  - $\diamond$  P1 : Smith was born on 09-Jan-55 is true in the current state of the world (i.e., of the database)
  - $\diamond$  P2 : Smith is female is false
- Compound propositions:
  - $\diamond~P1 \wedge P2 =$  Smith was born on 09-JAN-55  $\wedge$  Smith is female is false
  - $\Diamond \neg P2 =$  Smith is not female is true
- Much of the problem with the intuition of logic comes from implication, namely, with the fact that  $P \to Q$  is true when P is false

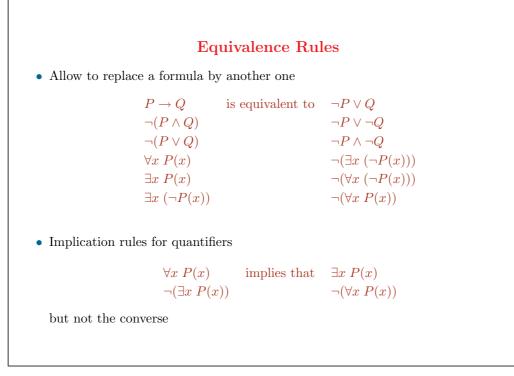




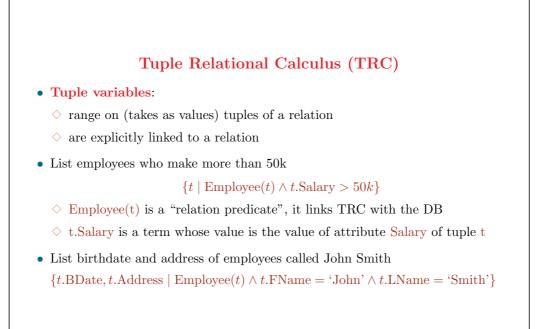


- $\forall$  (for all) and  $\exists$  (there exists)
- if you cannot do everything ...
  - $\diamond\,$  that does not mean that there is not anything that you can do ...
  - $\diamond\,$  nor that there is anything that you cannot do ...

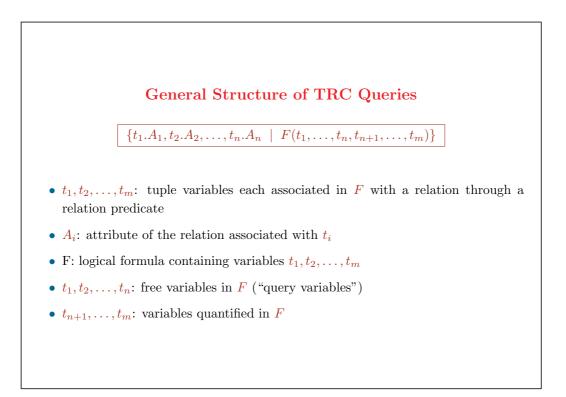




- 11
- This is about all the logic that is needed to master languages of traditional relational systems







# **TRC Semantics**

- F is evaluated for all possible values  $t_1, t_2, \ldots, t_n$  (= Cartesian product)
- If F is true for a tuple, then the projection  $t_1.A_1, t_2.A_2, \ldots, t_n.A_n$  is included in the result
- Result = nameless relation with n attributes; rules must be specified for deciding attribute names (e.g.,  $A_i$ 's if they are all distinct)

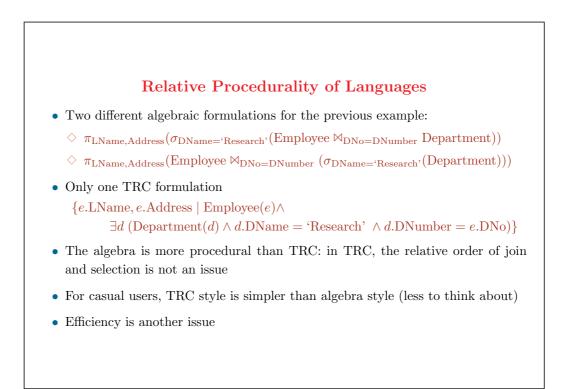
#### Structure of TRC Formulas

- Formula F is defined with the recursive structure of first-order logic
  - $\diamond R(t_i)$ , where R is a relation name
  - $\diamond t_i.A$  comparison  $t_j.B$
  - $\diamond t_i.A$  comparison constant
  - $\diamondsuit \ \neg F$
  - $\Diamond F_1 \wedge F_2$
  - $\diamond F_1 \lor F_2$
  - $\Diamond F_1 \to F_2$
  - $\Diamond F_1 \leftrightarrow F_2$
  - $\diamond \exists t \ F(t)$
  - $\diamond \ \forall t \ F(t)$
- Comparison: =,  $\neq$ , <, >,  $\leq$ ,  $\geq$

## Join

• List name and address of employees who work for the Research department

• "Join term" d.DNumber = e.DNo expresses a join between relation Department and relation Employee



- Efficiency:
  - $\diamond\,$  in most cases, the strategy that evaluates selection before joins is more efficient
  - $\diamond$  this is taken care of by the query optimizer of the DBMS

# **Two Joins**

• For every project located in Brussels, list the project number, the controling department number, and the name of the department manager

```
 \begin{aligned} \{p.\text{PNumber}, p.\text{DNum}, m.\text{LName} \mid \text{Project}(p) \land \\ \text{Employee}(m) \land p.\text{Location} = \text{`Brussels'} \land \\ \exists d \text{ (Department}(d) \land d.\text{DNumber} = p.\text{DNum} \land d.\text{MgrSSN} = m.\text{SSN}) \end{aligned}
```

• Same conclusion about procedurality: algebra is more procedural

17

• In this example, if *p*.DNum is replaced by *d*.DNumber in the target of the query, then the quantifier  $\exists d$  disappears, yielding a more symmetric formulation

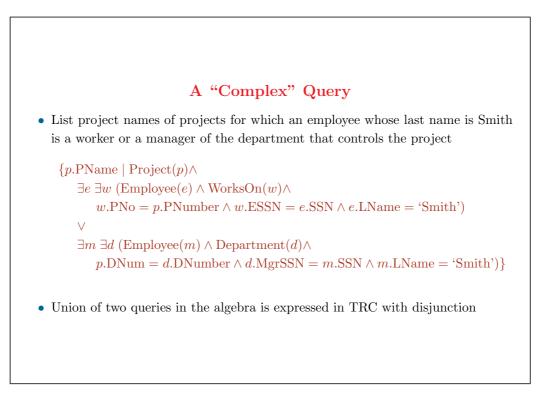
 $\{p. \text{PNumber}, d. \text{DNumber}, m. \text{LName} \mid \\ \text{Project}(p) \land \text{Employee}(m) \land \text{Department}(d) \land \\ p. \text{Location} = \text{Brussels} \land d. \text{DNumber} = p. \text{DNum} \land d. \text{MgrSSN} = m. \text{SSN} \}$ 

## Other Example with two Joins

• List the name of employees who work on some project controled by department number 5

 $\begin{aligned} & \{e.\text{FName}, e.\text{LName} \mid \text{Employee}(e) \land \\ & \exists p \; \exists w \; (\text{Project}(p) \land \text{WorksOn}(w) \land \\ & p.\text{DNum} = 5 \land w.\text{ESSN} = e.\text{SSN} \land p.\text{PNumber} = w.\text{PNo}) \end{aligned}$ 

• Same conclusion about procedurality: algebra is more procedural



- $\{x \mid P(x) \lor Q(x)\} \equiv \{x \mid P(x)\} \cup \{x \mid Q(x)\}$
- Other version: factor out of the disjunction the repeated

 $\exists e \; (\text{Employee}(e) \land e.\text{LName} = \text{Smith})$ 

# Join of a Relation with Itself

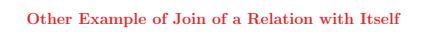
• List the first and last name of each employee, and the first and last name of his/her immediate supervisor

 $\{ e. \text{FName}, e. \text{LName}, s. \text{FName}, s. \text{LName} \mid \\ \text{Employee}(e) \land \text{Employee}(s) \land e. \text{SuperSSN} = s. \text{SSN} \}$ 

• The attributes of the result relation have to be specified explicitly (if the result is to be used elsewhere, i.e., not just displayed) through some kind of assignment

 $F(EmpFN, EmpLN, MgrFN, MgrLN) \leftarrow \{...\}$ 

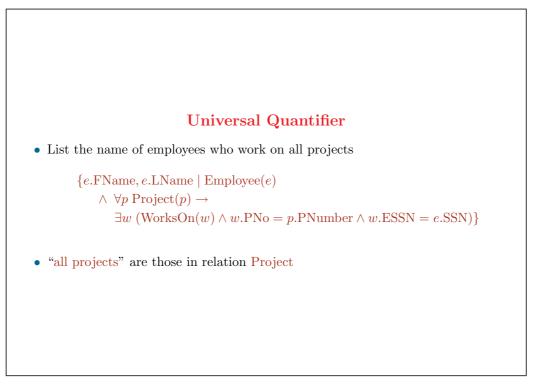
• Syntax is more difficult for the algebra, unless attributes are ordered



• List the SSN of employees who have both a dependent son and a dependent daughter

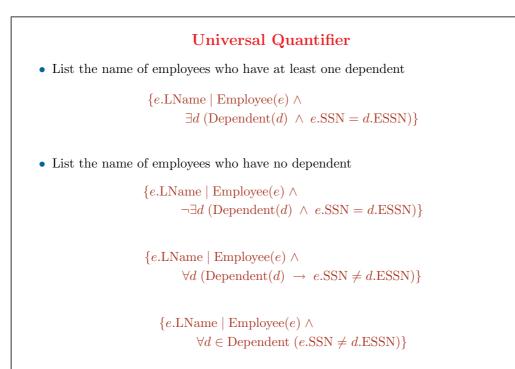
 $\{e.\text{ESSN} \mid \text{Dependent}(e) \\ \land \exists d \text{ (Dependent}(d) \\ \land e.\text{ESSN} = d.\text{ESSN} \\ \land d.\text{Relationship} = \text{`Son'} \\ \land d.\text{Relationship} = \text{`Daughter')} \}$ 



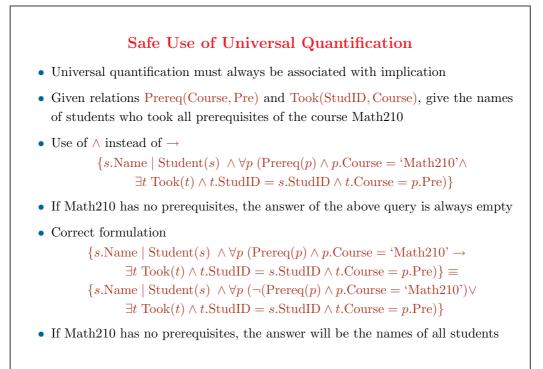


- Various styles of universal quantification (for List the employees who work on all projects):

  - ♦ logic with range-coupled quantifiers: { $e \in \text{Employee} \mid \forall p \in \text{Project (Workson(e,p))}$ }
  - ◇ towards natural language (where quantification is "infix" rather than "prefix" as in logic, binary predicates are also infix rather than prefix, and variables are seldom used as such):
    - \*  $\{e \in \text{Employee} \mid \text{for all } p \in \text{Project} (e \text{ Workson } p)\}$
    - \*  $\{e \in \text{Employee} \mid e \text{ Workson}(all p \in \text{Project})\}$
    - \* {Employee Workson (all Project)}



- Proof of equivalence of the formulations of List the name of employees who have no dependent by applying the equivalence rules of logic:
  - $\diamond \neg (\exists d \ P(d)) \equiv \forall d \ (\neg P(d))$
  - $\Diamond \neg \exists d (\text{Dependent}(d) \land e.\text{SSN} = d.\text{ESSN})$
  - $\Diamond \forall d \neg (\text{Dependent}(d) \land e.\text{SSN} = d.\text{ESSN})$
  - $\diamond \forall d \; (\neg \text{Dependent}(d) \lor \neg (e.\text{SSN} = d.\text{ESSN}))$
  - $\diamond \ \forall d \ (\neg \text{Dependent}(d) \lor e.\text{SSN} \neq d.\text{ESSN})$
  - $\Diamond \forall d \text{ (Dependent}(d) \rightarrow e.\text{SSN} \neq d.\text{ESSN})$
  - $\Diamond \forall d \in \text{Dependent} (e.\text{SSN} \neq d.\text{ESSN})$



# Safe TRC

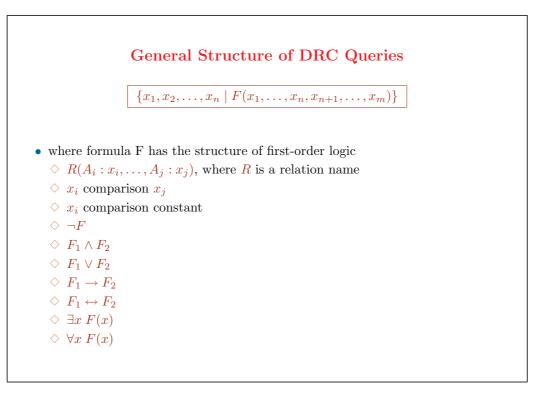
- Formulas with quantifiers, negation, some comparisons must be restricted so at to be meaningful
- Examples of ill-formed formulas with a comparison, a negation
  - $\diamond \ \{n \mid n \geq 3\}$
  - $\diamond \{e \mid \neg \text{Employee}(e)\}$
- Existential quantifiers
  - $\Diamond \exists t F(t)$  must have the form  $\exists t R(t) \land F'(t)$
  - $\diamond$  other notation:  $(\exists t \in R) F'(t)$
- Universal quantifiers must always be associated with implication
  - $\diamond \forall t \ F(t)$  must have the form  $\forall t \ R(t) \rightarrow F'(t)$
  - $\diamond$  other notation:  $(\forall t \in R) \ F'(t)$

- $(\exists t \in R)$  and  $(\forall t \in R)$  are called **range-restricted** or **ranged-coupled** quantifiers, where R is a relation predicate that defines and restricts the range of t
- General form of safe use of universal quantifier:  $\forall t \in (R(t) \land F'(t)) F''(t) (F'(t) \text{ and } F''(t))$  are any TRC formulas)
- Intuition:  $\forall t \ F(t)$ , where F(t) is a conjunction of database or comparison predicates, is meaningless (e.g.,  $\forall t \ Employee(t)$ )

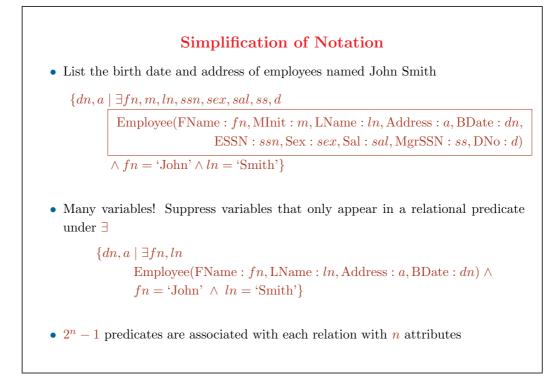


- **Domain variables** range on (i.e., take as values elements of) DB domains
- Relations are preferably viewed as predicates expressing properties of objects, represented as values
- **Relation predicates** (extensional predicates)
  - $\diamond\,$  realize the link between DRC and the DB
  - $\land R(A_1:x_1,\ldots,A_n:x_n)$  is associated with relation  $R(A_1:D_1,\ldots,A_n:D_n)$
  - $\diamond\ R(A_1:a_1,\ldots,A_n:a_n)$  is true if tuple  $\langle A_1:a_1,\ldots,A_n:a_n\rangle$  belongs to relation R

- Predicate WorksOn(ESSN:123456789, PNo:1, Hours:32.5) is true because tuple (ESSN:123456789, PNo:1, Hours:32.5) belongs to relation WorksOn
- In WorksOn(ESSN:123456789, PNo:1, Hours:32.5):
  - $\diamond$  WorksOn(ESSN: , PNo: , Hours: ) is the predicate name
  - $\diamond~$  123456789, 1 and 32.5 are the arguments



- 27
- As for TRC, the only things specific to DRC are the choice of domain variables and the definition of the relational predicates
- DRC has the structure of logic, applied as a DB query/assertion language
- Restrictions for safety similar to those of TRC for quantified formulas apply to DRC





# **Further Simplification**

• Suppress variables that only appear in a relation predicate and in a test for equality with a constant in a conjunction  $(\wedge)$ 

 $\{dn, a \mid \text{Employee}(\text{FName} : 'John', \text{LName} : 'Smith', \text{Address} : a, \text{BDate} : dn) \}$ 

- Corresponds to projection + selection on equality in the algebra
- The rest of DRC has the structure of logic

- $P(x) \wedge x = 3 \equiv P(3)$
- TRC formulation of the same example: {t.BDate, t.Address | Employee(t)  $\land t$ .FName = John  $\land t$ .LName = Smith}

# Selection + Projection

List the name of employees with a salary greater than  $50\mathrm{k}$ 

 $\{ fn, ln \mid \exists sal \\ (\text{Employee}(\text{FName} : fn, \text{LName} : ln, \text{Salary} : sal) \land sal > 50k) \}$ 

Could also conceivably be written

 $\{fn, ln \mid \text{Employee}(\text{FName} : fn, \text{LName} : ln, \text{Salary} :> 50k)\}$ 

# Join

• List name and address of employees who work in the Research department

 $\{fn, ln, a \mid \exists d \ (\text{Employee}(\text{FName} : fn, \text{LName} : ln, \text{Address} : a, \text{DNo} : d) \land \text{Department}(\text{DName} : '\text{Research'}, \text{DNumber} : d))\}$ 

- A join is expressed through the occurrence of the same domain variable in two (or more) relation predicates in a conjunction (∧)
- In TRC, a join is signaled by an explicit "join condition"

 $\{e.FName, e.LName, e.Address \mid Employee(e) \land \\ \exists d \ (Department(d) \ \land d.DName = `Research' \land d.DNumber = e.DNo) \}$ 

#### 31

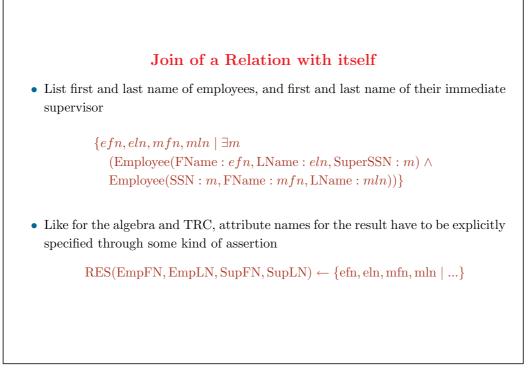
# Double Join • For every project located in Brussels, list the project number, the controling department number, and the name of the department manager $\{pn, d, mfn, mln \mid \exists e$ $\{Project(PNumber : pn, PLocation : 'Brussels', DNum : d) \land$ Department(MgrSSN : e, DNumber : d) $\land$ Employee(SSN : e, FName : mfn, LName : mln))}



• List project number of projects for which an employee whose last name is Smith is a worker or a manager of the department that controls the project

 $\{p \mid \text{Project}(\text{PNumber} : p) \land \exists e \text{ Employee}(\text{SSN} : e, \text{LName} : '\text{Smith'}) \land \\ [ WorksOn(\text{ESSN} : e, \text{PNo} : p) \lor \\ \exists d \text{ (Department}(\text{MgrSSN} : e, \text{DNumber} : d) \land \\ \text{Project}(\text{PNumber} : p, \text{DNum} : d)) ] \}$ 

• Many variants



# Universal Quantifier

• List the name of employees who work on all projects

```
\begin{split} \{fn, ln \mid \\ \exists e \text{ Employee}(\text{FName} : fn, \text{LName} : ln, \text{SSN} : e) \land \\ \forall p \text{ (Project}(\text{PNumber} : p) \rightarrow \text{WorksOn}(\text{PNo} : p, \text{ESSN} : e)) \rbrace \end{split}
```

 $\{fn, ln \mid$ 

```
\exists e \text{ Employee}(\text{FName} : fn, \text{LName} : ln, \text{SSN} : e) \land \\ \forall p \text{ (WorksOn}(\text{PNo} : p) \rightarrow \text{WorksOn}(\text{PNo} : p, \text{ESSN} : e)) \}
```

