

# Module 6

Active database

Deductive database

Temporal database

**Dr. Geetha Mary A**  
**Associate Professor ,**  
**SCOPE, Vellore Institute of Technology,**  
**Vellore**

**Source:**

**Pearson Education, Inc. 2011, Elmasri/Navathe, Fundamentals of Database Systems, Sixth Edition**

# Outline

- Active database & triggers
- Temporal databases
- Deductive databases

# Active Database Concepts and Triggers

## Generalized Model for Active Databases and Oracle **Triggers**

- **Triggers** are executed when a specified condition occurs during insert/delete/update
  - Triggers are action that fire automatically based on these conditions

# Event-Condition-Action (ECA) Model

## Generalized Model (contd.)

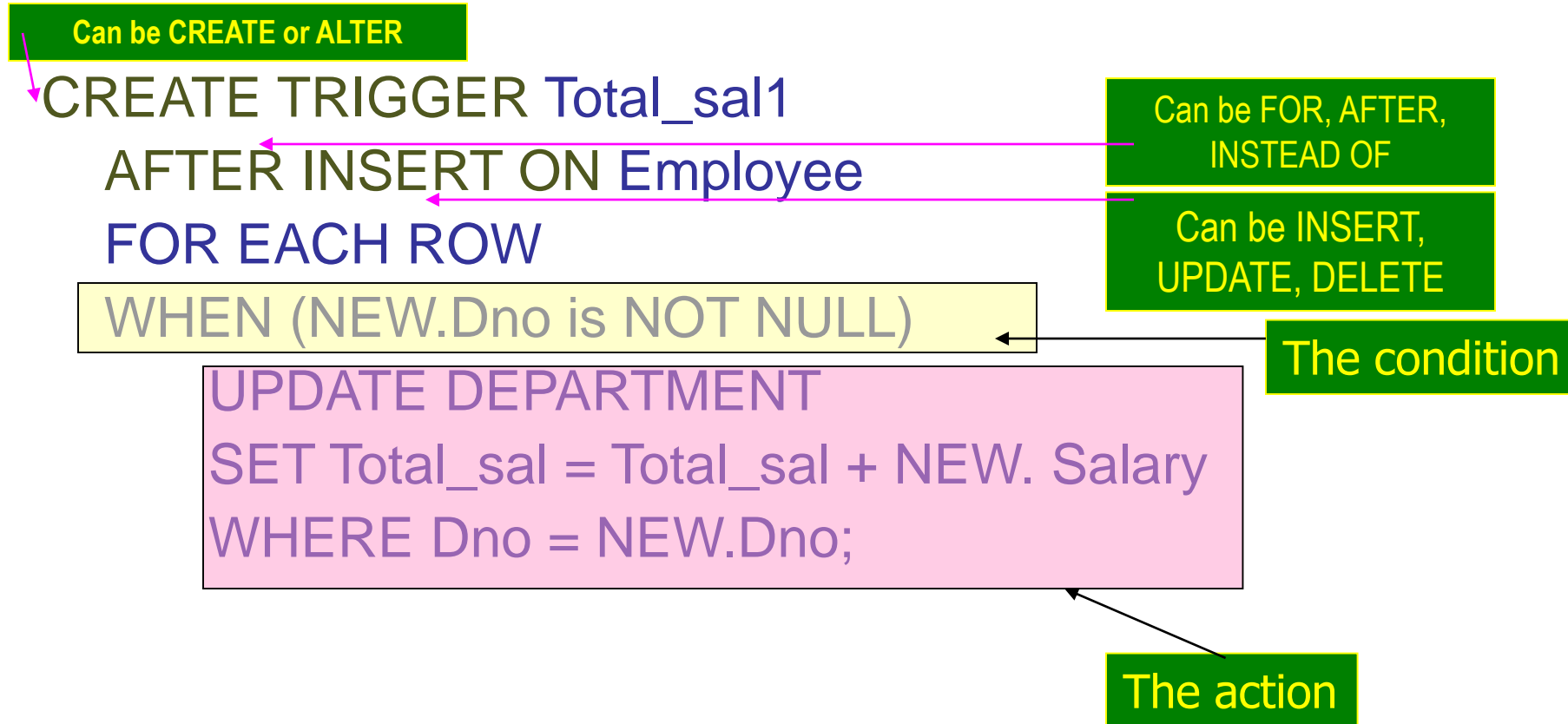
- Triggers follow an Event-condition-action (ECA) model
  - **Event:**
    - Database modification
      - E.g., insert, delete, update,
  - **Condition:**
    - Any true/false expression
      - Optional: If no condition is specified then condition is always true
  - **Action:**
    - Sequence of SQL statements that will be automatically executed

# Trigger Example

## Generalized Model (contd.)

- When a new employees is added to a department, modify the Total\_sal of the Department to include the new employees salary
- Logically this means that we will CREATE a **Condition** TRIGGER, let us call the trigger Total\_sal1
  - This trigger will execute AFTER INSERT ON Employee table
  - It will do the following FOR EACH ROW
    - WHEN NEW.Dno is NOT NULL
    - The trigger will UPDATE DEPARTMENT
    - By SETting the new Total\_sal to be the sum of
      - old Total\_sal and NEW. Salary
      - WHERE the Dno matches the NEW.Dno;

# Example: Trigger Definition



# CREATE or ALTER TRIGGER

## Generalized Model (contd.)

- CREATE TRIGGER <name>
  - Creates a trigger
- ALTER TRIGGER <name>
  - Alters a trigger (assuming one exists)
- CREATE OR ALTER TRIGGER <name>
  - Creates a trigger if one does not exist
  - Alters a trigger if one does exist
    - Works in both cases, whether a trigger exists or not

# Conditions

## Generalized Model (contd.)

- AFTER
  - Executes after the event
- BEFORE
  - Executes before the event
- INSTEAD OF
  - Executes **instead of** the event
    - Note that event does not execute in this case
      - E.g., used for modifying views



# Row-Level versus Statement-level

## Generalized Model (contd.)

- Triggers can be
  - **Row-level**
    - FOR EACH ROW specifies a row-level trigger
  - **Statement-level**
    - Default (when FOR EACH ROW is not specified)
- Row level triggers
  - Executed separately for each affected row
- Statement-level triggers
  - Execute once for the SQL statement,

# Condition

## Generalized Model (contd.)

- Any true/false condition to control whether a trigger is activated or not
  - Absence of condition means that the trigger will always execute for the event
  - Otherwise, condition is evaluated
    - before the event for BEFORE trigger
    - after the event for AFTER trigger

# Action

## Generalized Model (contd.)

- Action can be
  - One SQL statement
  - A sequence of SQL statements enclosed between a BEGIN and an END
- Action specifies the relevant modifications

# Triggers on Views

## Generalized Model (contd.)

- **INSTEAD OF** triggers are used to process view modifications

# Active Database Concepts and Triggers

## Design and Implementation Issues for Active Databases

- An active database allows users to make the following changes to triggers (rules)
  - Activate
  - Deactivate
  - Drop

# Active Database Concepts and Triggers

## Design and Implementation Issues for Active Databases

- An event can be considered in 3 ways
  - Immediate consideration
  - Deferred consideration
  - Detached consideration

# Active Database Concepts and Triggers

## Design and Implementation Issues (contd.)

- Immediate consideration
  - Part of the same transaction and can be one of the following depending on the situation
    - Before
    - After
    - Instead of
- Deferred consideration
  - Condition is evaluated at the end of the transaction
- Detached consideration
  - Condition is evaluated in a separate transaction

# Active Database Concepts and Triggers

## Potential Applications for Active Databases

### ■ Notification

- Automatic notification when certain condition occurs
- Enforcing integrity constraints
  - Triggers are smarter and more powerful than constraints
- Maintenance of derived data
  - Automatically update derived data and avoid anomalies due to redundancy
    - E.g., trigger to update the Total\_sal in the earlier example



# Active Database Concepts and Triggers

## Triggers in SQL-99

- Can alias variables inside the REFERENCING clause

# Active Database Concepts and Triggers

## ■ Trigger examples

```
T1:  CREATE TRIGGER Total_sal1
      AFTER UPDATE OF Salary ON EMPLOYEE
      REFERENCING OLD ROW AS O, NEW ROW AS N
      FOR EACH ROW
      WHEN ( N.Dno IS NOT NULL )
      UPDATE DEPARTMENT
      SET Total_sal = Total_sal + N.salary - O.salary
      WHERE Dno = N.Dno;

T2:  CREATE TRIGGER Total_sal2
      AFTER UPDATE OF Salary ON EMPLOYEE
      REFERENCING OLD TABLE AS O, NEW TABLE AS N
      FOR EACH STATEMENT
      WHEN EXISTS ( SELECT * FROM N WHERE N.Dno IS NOT NULL ) OR
                EXISTS ( SELECT * FROM O WHERE O.Dno IS NOT NULL )
      UPDATE DEPARTMENT AS D
      SET D.Total_sal = D.Total_sal
      + ( SELECT SUM (N.Salary) FROM N WHERE D.Dno=N.Dno )
      - ( SELECT SUM (O.Salary) FROM O WHERE D.Dno=O.Dno )
      WHERE Dno IN ( ( SELECT Dno FROM N ) UNION ( SELECT Dno FROM O ) );
```

# Temporal Database Concepts

## Time Representation, Calendars, and Time Dimensions

- Time is considered **ordered sequence** of **points** in some **granularity**
  - Use the term **choronon** instead of point to describe minimum granularity

# Temporal Database Concepts

## Time Representation, ... (contd.)

- A **calendar** organizes time into different time units for convenience.
  - Accommodates various calendars
    - Gregorian (western)
    - Chinese
    - Islamic
    - Hindu
    - Jewish
    - Etc.

# Temporal Database Concepts

## Time Representation, ... (contd.)

- Point events
  - Single time point event
    - E.g., bank deposit
  - Series of point events can form a time series data
- Duration events
  - Associated with specific time period
    - Time period is represented by start time and end time

# Temporal Database Concepts

## Time Representation, ... (contd.)

- Transaction time
  - The time when the information from a certain transaction becomes valid
- Bitemporal database
  - Databases dealing with two time dimensions

# Temporal Database Concepts

## Incorporating Time in Relational Databases Using Tuple Versioning

- Add to every tuple
  - Valid start time
  - Valid end time

# Temporal Database Concepts

(a) EMP\_VT

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Vst</u>	Vet
------	------------	--------	-----	----------------	------------	-----

DEPT\_VT

Dname	<u>Dno</u>	Total_sal	Manager_ssn	<u>Vst</u>	Vet
-------	------------	-----------	-------------	------------	-----

(b) EMP\_TT

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Tst</u>	Tet
------	------------	--------	-----	----------------	------------	-----

DEPT\_TT

Dname	<u>Dno</u>	Total_sal	Manager_ssn	<u>Tst</u>	Tet
-------	------------	-----------	-------------	------------	-----

(c) EMP\_BT

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Vst</u>	Vet	<u>Tst</u>	Tet
------	------------	--------	-----	----------------	------------	-----	------------	-----

DEPT\_BT

Dname	<u>Dno</u>	Total_sal	Manager_ssn	<u>Vst</u>	Vet	<u>Tst</u>	Tet
-------	------------	-----------	-------------	------------	-----	------------	-----

**Figure 24.7**

Different types of temporal relational databases.  
(a) Valid time database schema. (b) Transaction time database schema. (c) Bitemporal database schema.



# Temporal Database Concepts

**Figure 24.8**

Some tuple versions in the valid time relations EMP\_VT and DEPT\_VT.

**EMP\_VT**

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Vst</u>	Vet
Smith	123456789	25000	5	333445555	2002-06-15	2003-05-31
Smith	123456789	30000	5	333445555	2003-06-01	Now
Wong	333445555	25000	4	999887777	1999-08-20	2001-01-31
Wong	333445555	30000	5	999887777	2001-02-01	2002-03-31
Wong	333445555	40000	5	888665555	2002-04-01	Now
Brown	222447777	28000	4	999887777	2001-05-01	2002-08-10
Narayan	666884444	38000	5	333445555	2003-08-01	Now

...

**DEPT\_VT**

Dname	<u>Dno</u>	Manager_ssn	<u>Vst</u>	Vet
Research	5	888665555	2001-09-20	2002-03-31
Research	5	333445555	2002-04-01	Now

# Temporal Database Concepts

## Incorporating Time in Object-Oriented Databases Using Attribute Versioning

- A single complex object stores all temporal changes of the object
- **Time varying attribute**
  - An attribute that changes over time
    - E.g., age
- **Non-Time varying attribute**
  - An attribute that does **not** changes over time
    - E.g., date of birth

# Introduction to Deductive Databases

- Overview of Deductive Databases
- Prolog/Datalog Notation
- Datalog Notation
- Clausal Form and Horn Clauses
- Interpretation of Rules
- Datalog Programs and Their Safety
- Use the Relational Operations
- Evaluation of Non-recursive Datalog Queries

# Overview of Deductive Databases

- **Declarative Language**
  - Language to specify rules
- **Inference Engine (Deduction Machine)**
  - Can deduce new facts by interpreting the rules
  - Related to logic programming
    - Prolog language (Prolog => **P**rogramming in **l**ogic)
    - Uses backward chaining to evaluate
      - Top-down application of the rules

# Overview of Deductive Databases

- Speciation consists of:
  - Facts
    - Similar to relation specification without the necessity of including attribute names
  - Rules
    - Similar to relational views (virtual relations that are not stored)

# Prolog/Datalog Notation

- **Predicate** has
  - a name
  - a fixed number of arguments
    - **Convention:**
      - Constants are numeric or character strings
      - Variables start with upper case letters
  - E.g., **SUPERVISE(Supervisor, Supervisee)**
    - States that Supervisor **SUPERVISE(s)** Supervisee

# Prolog/Datalog Notation

## ■ Rule

- Is of the form head :- body
  - where :- is read as if and only iff
- E.g., SUPERIOR(X,Y) :- SUPERVISE(X,Y)
- E.g., SUBORDINATE(Y,X) :- SUPERVISE(X,Y)

# Prolog/Datalog Notation

## ■ Query

- Involves a predicate symbol followed by y some variable arguments to answer the question
  - where :- is read as if and only iff
- E.g., SUPERIOR(james,Y)?
- E.g., SUBORDINATE(james,X)?



# Figure 24.11

## ■ (a) Prolog notation (b) Supervisory tree

(a)

### Facts

```
SUPERVISE(franksin, john).  
SUPERVISE(franksin, ramesh).  
SUPERVISE(franksin, joyce).  
SUPERVISE(jennifer, alicia).  
SUPERVISE(jennifer, ahmad).  
SUPERVISE(james, franklin).  
SUPERVISE(james, jennifer).  
...
```

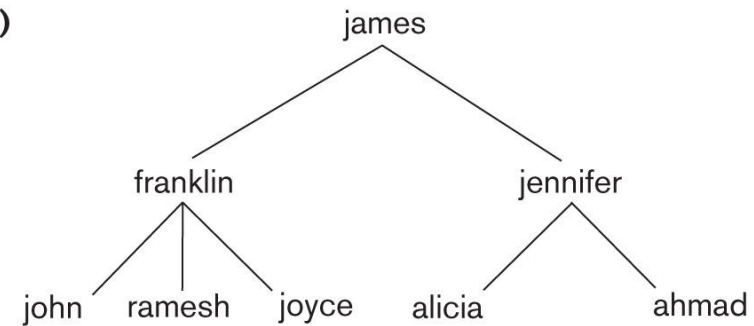
### Rules

```
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)?
```

(b)



# Datalog Notation

- Datalog notation
  - Program is built from **atomic formulae**
    - **Literals** of the form  $p(a_1, a_2, \dots, a_n)$  where
      - $p$  predicate name
      - $n$  is the number of arguments
    - Built-in predicates are included
      - E.g.,  $<$ ,  $<=$ , etc.
  - A **literal** is either
    - An atomic formula
    - An atomic formula preceded by not

# Clausal Form and Horn Clauses

- A formula can have quantifiers
  - Universal
  - Existential

# Clausal Form and Horn Clauses

- In clausal form, a formula must be transformed into another formula with the following characteristics
  - All variables are universally quantified
  - Formula is made of a number of clauses where each clause is made up of literals connected by logical ORs only
  - Clauses themselves are connected by logical ANDs only.

# Clausal Form and Horn Clauses

- Any formula can be converted into a clausal form
  - A specialized case of clausal form are horn clauses that can contain no more than one positive literal
- Datalog program are made up of horn clauses

# Interpretation of Rules

- There are two main alternatives for interpreting rules:
  - **Proof-theoretic**
  - **Model-theoretic**

# Interpretation of Rules

- Proof-theoretic

- Facts and rules are **axioms**
- **Ground axioms** contain no variables
- Rules are **deductive axioms**
- **Deductive axioms** can be used to construct new facts from existing facts
- This process is known as theorem proving

# Proving a new fact

## ■ Figure 24.12

1.  $\text{SUPERIOR}(X, Y) \text{ :- SUPERVISE}(X, Y).$  (rule 1)
2.  $\text{SUPERIOR}(X, Y) \text{ :- SUPERVISE}(X, Z), \text{SUPERIOR}(Z, Y).$  (rule 2)
3.  $\text{SUPERVISE}(\text{jennifer}, \text{ahmad}).$  (ground axiom, given)
4.  $\text{SUPERVISE}(\text{james}, \text{jennifer}).$  (ground axiom, given)
5.  $\text{SUPERIOR}(\text{jennifer}, \text{ahmad}).$  (apply rule 1 on 3)
6.  $\text{SUPERIOR}(\text{james}, \text{ahmad}).$  (apply rule 2 on 4 and 5)



# Interpretation of Rules

- Model-theoretic

- Given a finite or infinite domain of constant values, we assign the predicate every combination of values as arguments
- If this is done for every predicated, it is called **interpretation**

# Interpretation of Rules

- **Model**

- An **interpretation** for a specific set of rules

- **Model-theoretic proofs**

- Whenever a particular substitution to the variables in the rules is applied, if all the predicated are true under the interpretation, the predicate at the head of the rule must also be true

- **Minimal model**

- Cannot change any fact from true to false and still get a model for these rules

# Minimal model

## ■ Figure 24.13

### Rules

$\text{SUPERIOR}(X, Y) :- \text{SUPERVISE}(X, Y).$

$\text{SUPERIOR}(X, Y) :- \text{SUPERVISE}(X, Z), \text{SUPERIOR}(Z, Y).$

### Interpretation

#### *Known Facts:*

$\text{SUPERVISE}(\text{franklin}, \text{john})$  is **true**.

$\text{SUPERVISE}(\text{franklin}, \text{ramesh})$  is **true**.

$\text{SUPERVISE}(\text{franklin}, \text{joyce})$  is **true**.

$\text{SUPERVISE}(\text{jennifer}, \text{alicia})$  is **true**.

$\text{SUPERVISE}(\text{jennifer}, \text{ahmad})$  is **true**.

$\text{SUPERVISE}(\text{james}, \text{franklin})$  is **true**.

$\text{SUPERVISE}(\text{james}, \text{jennifer})$  is **true**.

$\text{SUPERVISE}(X, Y)$  is **false** for all other possible  $(X, Y)$  combinations

#### *Derived Facts:*

$\text{SUPERIOR}(\text{franklin}, \text{john})$  is **true**.

$\text{SUPERIOR}(\text{franklin}, \text{ramesh})$  is **true**.

$\text{SUPERIOR}(\text{franklin}, \text{joyce})$  is **true**.

$\text{SUPERIOR}(\text{jennifer}, \text{alicia})$  is **true**.

$\text{SUPERIOR}(\text{jennifer}, \text{ahmad})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{franklin})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{jennifer})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{john})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{ramesh})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{joyce})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{alicia})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{ahmad})$  is **true**.

$\text{SUPERIOR}(X, Y)$  is **false** for all other possible  $(X, Y)$  combinations

EMPLOYEE(john).  
 EMPLOYEE(frunklin).  
 EMPLOYEE(alicia).  
 EMPLOYEE(jennifer).  
 EMPLOYEE(ramesh).  
 EMPLOYEE(joyce).  
 EMPLOYEE(ahmad).  
 EMPLOYEE(james).  
  
 SALARY(john, 30000).  
 SALARY(frunklin, 40000).  
 SALARY(alicia, 25000).  
 SALARY(jennifer, 43000).  
 SALARY(ramesh, 38000).  
 SALARY(joyce, 25000).  
 SALARY(ahmad, 25000).  
 SALARY(james, 55000).  
  
 DEPARTMENT(john, research).  
 DEPARTMENT(frunklin, research).  
 DEPARTMENT(alicia, administration).  
 DEPARTMENT(jennifer, administration).  
 DEPARTMENT(ramesh, research).  
 DEPARTMENT(joyce, research).  
 DEPARTMENT(ahmad, administration).  
 DEPARTMENT(james, headquarters).  
  
 SUPERVISE(frunklin, john).  
 SUPERVISE(frunklin, ramesh).  
 SUPERVISE(frunklin, joyce).  
 SUPERVISE(jennifer, alicia).  
 SUPERVISE(jennifer, ahmad).  
 SUPERVISE(james, frunklin).  
 SUPERVISE(james, jennifer).

MALE(john).  
 MALE(frunklin).  
 MALE(ramesh).  
 MALE(ahmad).  
 MALE(james).  
  
 FEMALE(alicia).  
 FEMALE(jennifer).  
 FEMALE(joyce).  
  
 PROJECT(productx).  
 PROJECT(producty).  
 PROJECT(productz).  
 PROJECT(computerization).  
 PROJECT(reorganization).  
 PROJECT(newbenefits).  
  
 WORKS\_ON(john, productx, 32).  
 WORKS\_ON(john, producty, 8).  
 WORKS\_ON(ramesh, productz, 40).  
 WORKS\_ON(joyce, productx, 20).  
 WORKS\_ON(joyce, producty, 20).  
 WORKS\_ON(frunklin, producty, 10).  
 WORKS\_ON(frunklin, productz, 10).  
 WORKS\_ON(frunklin, computerization, 10).  
 WORKS\_ON(frunklin, reorganization, 10).  
 WORKS\_ON(alicia, newbenefits, 30).  
 WORKS\_ON(alicia, computerization, 10).  
 WORKS\_ON(ahmad, computerization, 35).  
 WORKS\_ON(ahmad, newbenefits, 5).  
 WORKS\_ON(jennifer, newbenefits, 20).  
 WORKS\_ON(jennifer, reorganization, 15).  
 WORKS\_ON(james, reorganization, 10).

# Datalog Programs and Their Safety

- Two main methods of defining truth values
  - Fact-defined predicates (or relations)
    - Listing all combination of values that make a predicate true
  - Rule-defined predicates (or views)
    - Head (LHS) of 1 or more Datalog rules, for example Figure 24.15

```
SUPERIOR(X, Y) :- SUPERVISE(X, Y).  
SUPERIOR(X, Y) :- SUPERVISE(X, Z), SUPERIOR(Z, Y).
```

```
SUBORDINATE(X, Y) :- SUPERIOR(Y, X).
```

```
SUPERVISOR(X) :- EMPLOYEE(X), SUPERVISE(X, Y).
```

```
OVER_40K_EMP(X) :- EMPLOYEE(X), SALARY(X, Y), Y >= 40000.  
UNDER_40K_SUPERVISOR(X) :- SUPERVISOR(X), NOT(OVER_40K_EMP(X)).  
MAIN_PRODUCTX_EMP(X) :- EMPLOYEE(X), WORKS_ON(X, productx, Y), Y >= 20.  
PRESIDENT(X) :- EMPLOYEE(X), NOT(SUPERVISE(Y, X)).
```

# Datalog Programs and Their Safety

- A program is **safe** if it generates a **finite** set of facts
  - Fact-defined predicates (or relations)
    - Listing all combination of values that make a predicate true
  - Rule-defined predicates (or views)
    - Head (LHS) of 1 or more Datalog rules, for example Figure 24.15

# Use the Relational Operations

- Many operations of relational algebra can be defined in the form of Datalog rules that define the result of applying these operations on database relations (fact predicates)
  - Relational queries and views can be easily specified in Datalog

# Evaluation of Non-recursive Datalog Queries

- Define an inference mechanism based on relational database query processing concepts
  - See Figure 24.17 on predicate dependencies for Figs 24.14 and 24.15

