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

Condition

- Logically this means that we will CREATE a 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

Can be CREATE or ALTER

```
CREATE TRIGGER Total_sal1
    AFTER INSERT ON Employee
    FOR EACH ROW
    WHEN (NEW.Dno is NOT NULL)
        UPDATE DEPARTMENT
        SET Total_sal = Total_sal + NEW. Salary
        WHERE Dno = NEW.Dno;
```

Can be FOR, AFTER, INSTEAD OF

Can be INSERT, UPDATE, DELETE

The condition

The action

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 on not
  - Absence of condition means that the trigger will always execute for the even
  - 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 databases require some aspect of time when organizing information
  - Healthcare
  - Insurance
  - Reservation systems
  - Scientific databases

SQL2 temporal data types

- DATE, TIME, TIMESTAMP, INTERVAL, PERIOD

Temporal Database Concepts

1) 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

2) A temporal database stores data relating to time instances. It offers temporal data types and stores information relating to past, present and future time. Temporal databases could be uni-temporal, bi-temporal or tri-temporal.

Uni-Temporal [edit]

A uni-temporal database has one axis of time, either the validity range or the system time range.

Bi-Temporal [edit]

A bi-temporal database has two axis of time:

- valid time
- transaction time or decision time

4)

Tri-Temporal [edit]

A tri-temporal database has three axes of time:

- valid time
- transaction time
- decision time

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.

3)

- **Valid time** is the time period during which a fact is true in the real world.
- **Transaction time** is the time at which a fact was recorded in the database.
- **Decision time** is the time at which the decision was made about the fact.

Features [edit]

Temporal databases support managing and accessing temporal data by providing one or more of the following features:[1][2]

- A time period datatype, including the ability to represent time periods with no end (infinity or forever)
- The ability to define valid and transaction time period attributes and bitemporal relations
- System-maintained transaction time
- Temporal primary keys, including non-overlapping period constraints
- Temporal constraints, including non-overlapping uniqueness and referential integrity
- Update and deletion of temporal records with automatic splitting and coalescing of time periods
- Temporal queries at current time, time points in the past or future, or over durations
- Predicates for querying time periods, often based on Allen's interval relations

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

VALID TIME
data-base schema

(a) **EMP_VT**

Name	Ssn	Salary	Dno	Supervisor_ssn	Vst	Vet

DEPT_VT

Dname	Dno	Total_sal	Manager_ssn	Vst	Vet

TRANSACTION TIME
data-base schema

(b) **EMP_TT**

Name	Ssn	Salary	Dno	Supervisor_ssn	Tst	Tet

DEPT_TT

Dname	Dno	Total_sal	Manager_ssn	Tst	Tet

BITEMPORAL
data-base schema

(c) **EMP_BT**

Name	Ssn	Salary	Dno	Supervisor_ssn	Vst	Vet	Tst	Tet

DEPT_BT

Dname	Dno	Total_sal	Manager_ssn	Vst	Vet	Tst	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	Ssn	Salary	Dno	Supervisor_ssn	Vst	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	Dno	Manager_ssn	Vst	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

- Types of updates
  - Proactive
  - Retroactive
  - Simultaneous
- Implementation considerations
  - Store all tuples in the same table
  - Create two tables: one for currently valid information and one for the rest
  - Vertically partition temporal relation attributes into separate relations
    - New tuple created whenever any attribute updated
- Append-only database
  - Keeps complete record of changes and corrections

```
class TEMPORAL_SALARY
{     attribute     Date                 Valid_start_time;
      attribute     Date                 Valid_end_time;
      attribute     float                Salary;
};

class TEMPORAL_DEPT
{     attribute     Date                 Valid_start_time;
      attribute     Date                 Valid_end_time;
      attribute     DEPARTMENT_VT        Dept;
};

class TEMPORAL_SUPERVISOR
{     attribute     Date                 Valid_start_time;
      attribute     Date                 Valid_end_time;
      attribute     EMPLOYEE_VT          Supervisor;
};

class TEMPORAL_LIFESPAN
{     attribute     Date                 Valid_ start time;
      attribute     Date                 Valid end time;
};

class EMPLOYEE_VT
(     extent EMPLOYEES )
{     attribute     list<TEMPORAL_LIFESPAN>       lifespan;
      attribute     string                        Name;
      attribute     string                        Ssn;
      attribute     list<TEMPORAL_SALARY>         Sal_history;
      attribute     list<TEMPORAL_DEPT>           Dept_history;
      attribute     list <TEMPORAL_SUPERVISOR>    Supervisor_history;
};
```

- CREATE TABLE statement
  - Extended with optional AS clause
  - Allows users to declare different temporal options
  - Examples:
    - AS VALID STATE<GRANULARITY> (valid time relation with valid time period)
    - AS TRANSACTION (transaction time relation with transaction time period)
- Keywords STATE and EVENT
  - Specify whether a time period or point associated with valid time dimension
- Time series data
  - Often used in financial, sales, and economics applications
  - Special type of valid event data
  - Event's time points predetermined according to fixed calendar
  - Managed using specialized time series management systems
  - Supported by some commercial DBMS packages

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

What is a deductive database system?

A deductive database can be defined as an advanced database augmented with an inference system.

By evaluating rules against facts, new facts can be derived, which in turn can be used to answer queries. It makes a database system more powerful.

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 => **Pro**gramming in **log**ic)
    - Uses backward chaining to evaluate
      - Top-down application of the rules

In a deductive database system we typically specify rules through a **declarative language**—a language in which we specify what to achieve rather than how to achieve it. An **inference engine** (or **deduction mechanism**) within the system can deduce new facts from the database by interpreting these rules. The model used for deductive databases is closely related to the relational data model, and particularly to the domain relational calculus formalism (see Section 6.6). It is also related to the field of **logic programming** and the**Prolog** language. The deductive database work based on logic has used Prolog as a starting point. A variation of Prolog called**Datalog** is used to define rules declaratively in conjunction with an existing set of relations, which are themselves treated as literals in the language. Although the language structure of Datalog resembles that of Prolog, its operational semantics—that is, how a Datalog program is executed—is still different.

Facts are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names. Recall that a tuple in a relation describes some real-world fact whose meaning is partly determined by the attribute names. In a deductive database, the meaning of an attribute value in a tuple is determined solely by its position within the tuple.

Rules are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms 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 basic relational views.

rules → recursion
view → ✗

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)

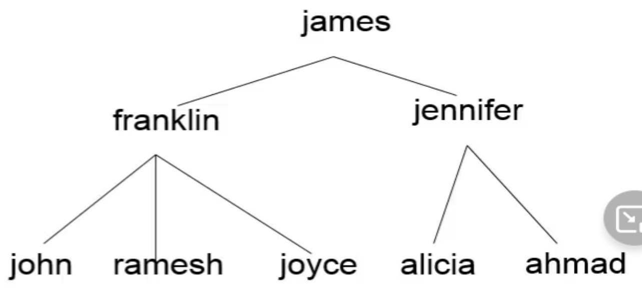
Example (a logic program)

Facts:

supervise(franklin, john),
supervise(franklin, ramesh),
supervise(franklin, joyce)
supervise(james, franklin),
supervise(jennifer, alicia),
supervise(jennifer, ahmad),
supervise(james, jennifer).

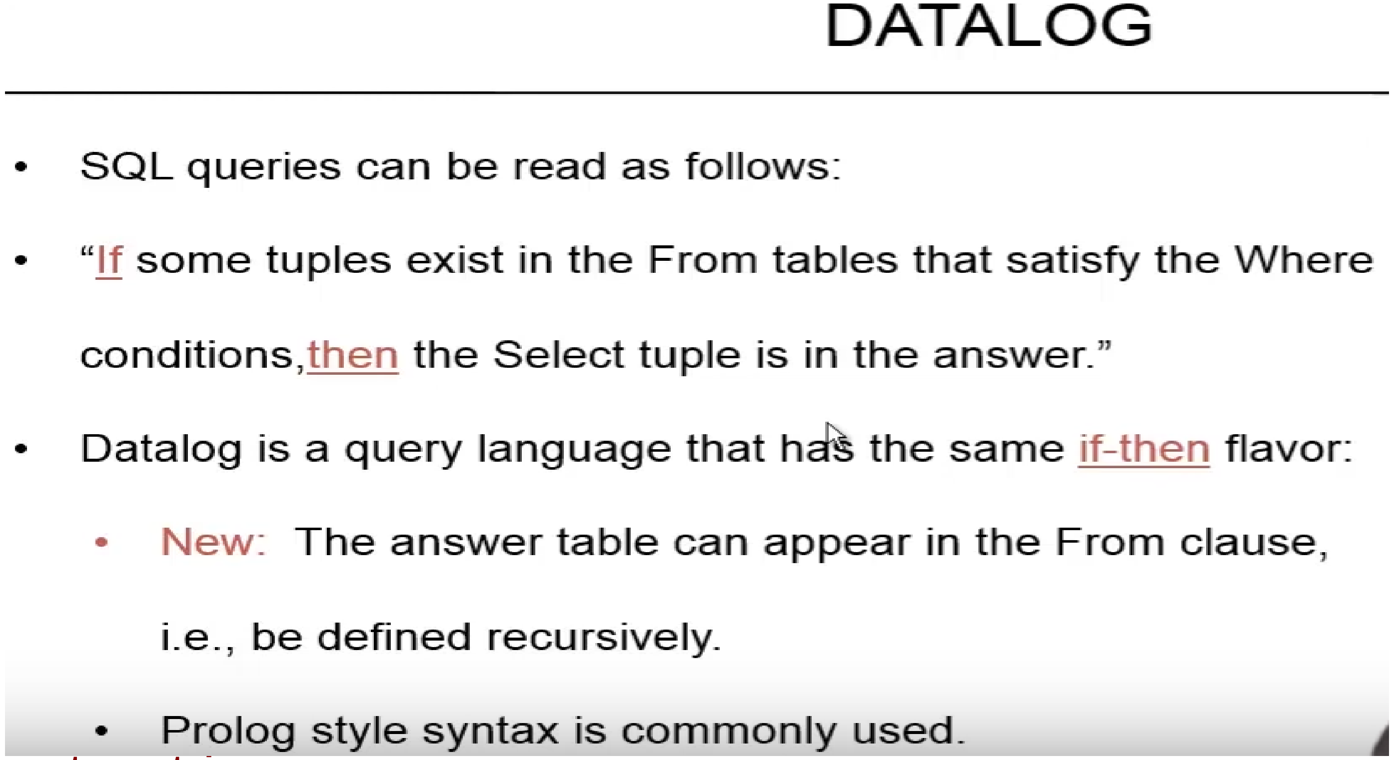
Rules:

superior(X, Y) ← supervise(X, Y),
superior(X, Y) ← supervise(X, Z), superior(Z, Y),
subordinary(X, Y) ← superior(Y, X).



Prolog/Datalog Notation

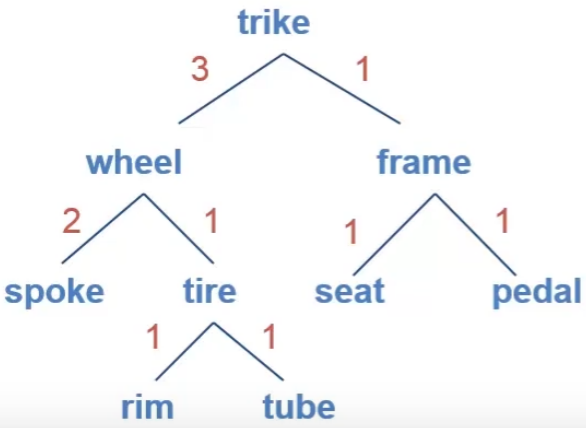
- **Predicate** has
  - a name
  - a fixed number of arguments
    - Convention:
      - Constants are numeric or character strings
    - Variables start with upper case letters
  - E.g., SUPERVISE(Supervisor, Supervisee)
    - States that Supervisor SUPERVISE(s) Supervisee



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)

Example



part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

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)?

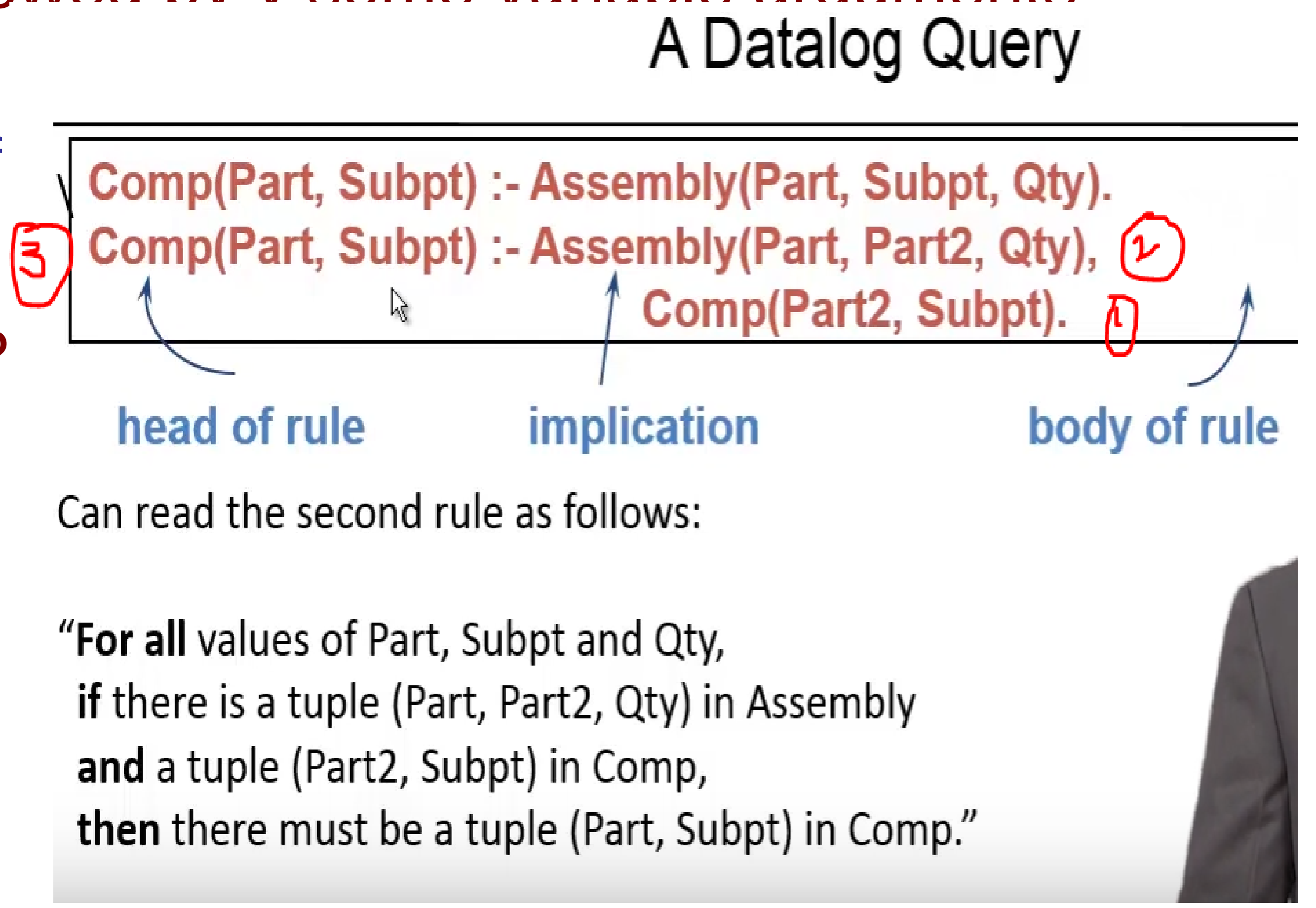


Figure 24.11

- (a) Prolog notation (b) Supervisory tree

(a)

Facts

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

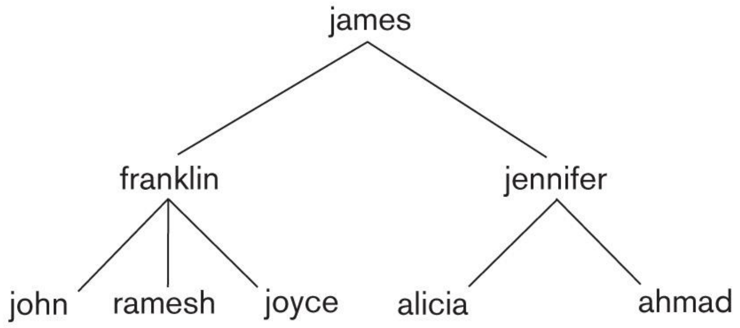
Rules

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(a1, a2, … an) 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

- Clause

A	B	$\neg A \vee B$
1	1	1
0	1	1
1	0	0
0	0	1

A	B	$B \leftarrow A$
1	1	1
0	1	1
1	0	0
0	0	1

- Horn clause

A Horn clause is a clause with the head containing only one positive atom.

$$B_m \leftarrow A_1, \ldots, A_n$$

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. SUPERIOR(X, Y) :– SUPERVISE(X, Y).	(rule 1)
2. SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).	(rule 2)
3. SUPERVISE(jennifer, ahmad).	(ground axiom, given)
4. SUPERVISE(james, jennifer).	(ground axiom, given)
5. SUPERIOR(jennifer, ahmad).	(apply rule 1 on 3)
6. SUPERIOR(james, 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
SUPERIOR(X, Y) :– SUPERVISE(X, Y).
SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).

Interpretation

Known Facts:
SUPERVISE(franklin, john) is **true**.
SUPERVISE(franklin, ramesh) is **true**.
SUPERVISE(franklin, joyce) is **true**.
SUPERVISE(jennifer, alicia) is **true**.
SUPERVISE(jennifer, ahmad) is **true**.
SUPERVISE(james, franklin) is **true**.
SUPERVISE(james, jennifer) is **true**.
SUPERVISE(X, Y) is **false** for all other possible (X, Y) combinations

Derived Facts:
SUPERIOR(franklin, john) is **true**.
SUPERIOR(franklin, ramesh) is **true**.
SUPERIOR(franklin, joyce) is **true**.
SUPERIOR(jennifer, alicia) is **true**.
SUPERIOR(jennifer, ahmad) is **true**.
SUPERIOR(james, franklin) is **true**.
SUPERIOR(james, jennifer) is **true**.
SUPERIOR(james, john) is **true**.
SUPERIOR(james, ramesh) is **true**.
SUPERIOR(james, joyce) is **true**.
SUPERIOR(james, alicia) is **true**.
SUPERIOR(james, ahmad) is **true**.
SUPERIOR(X, Y) is **false** for all other possible (X, Y) combinations

```
EMPLOYEE(john).
EMPLOYEE(franklin).
EMPLOYEE(alicia).
EMPLOYEE(jennifer).
EMPLOYEE(ramesh).
EMPLOYEE(joyce).
EMPLOYEE(ahmad).
EMPLOYEE(james).

SALARY(john, 30000).
SALARY(franklin, 40000).
SALARY(alicia, 25000).
SALARY(jennifer, 43000).
SALARY(ramesh, 38000).
SALARY(joyce, 25000).
SALARY(ahmad, 25000).
SALARY(james, 55000).

DEPARTMENT(john, research).
DEPARTMENT(franklin, research).
DEPARTMENT(alicia, administration).
DEPARTMENT(jennifer, administration).
DEPARTMENT(ramesh, research).
DEPARTMENT(joyce, research).
DEPARTMENT(ahmad, administration).
DEPARTMENT(james, headquarters).

SUPERVISE(frankl1n, john).
SUPERVISE(frankl1n, ramesh)
SUPERVISE(frankin , joyce).
SUPERVISE(jennifer, alicia).
SUPERVISE(jennifer, ahmad).
SUPERVISE(james, franklin).
SUPERVISE(james, jennifer).

MALE(john).
MALE(franklin).
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(franklin, producty, 10).
WORKS_ON(franklin, productz, 10).
WORKS_ON(franklin, computerization, 10).
WORKS_ON(franklin, 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_40_K_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 for of Datalog rules that defined 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