# Datalog

R&G Ch 24

# Dependency Preserving Decomposition

- Simple modification to the algorithm for 3NF:
  - If decomposition can't enforce $X \rightarrow Y$, add relation XY.
- But XY may still violate 3NF
  - E.g., Relation ABC with FDs $AB \rightarrow C$, $B \rightarrow C$
- **Refinement**: Only enforce FDs in the Minimal Cover.

# Minimal Cover

- For a set of FDs F, the minimal cover G satisfies:
  - Closure of F = Closure of G
  - RHS of each FD in G is a single attribute
  - Any deletion of an FD in G or attributes in an FD in G changes its closure.

# Example

For the following set of FDs

A→B  ABCD→E  EF→GH  ACDF→EG

A→B  ACD→E  EF→G  EF→H

… is the minimal cover

# Example

For the following set of FDs

A→B  ABCD→E  EF→GH  ACDF→EG



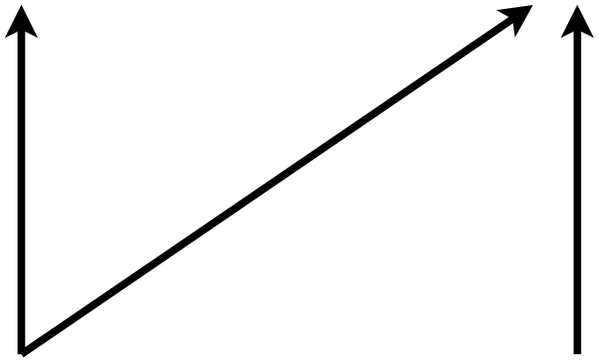
A→B  ACD→E  EF→G  EF→H

… is the minimal cover

# Example

For the following set of FDs

A→B  ABCD→E  EF→GH  ACDF→EG

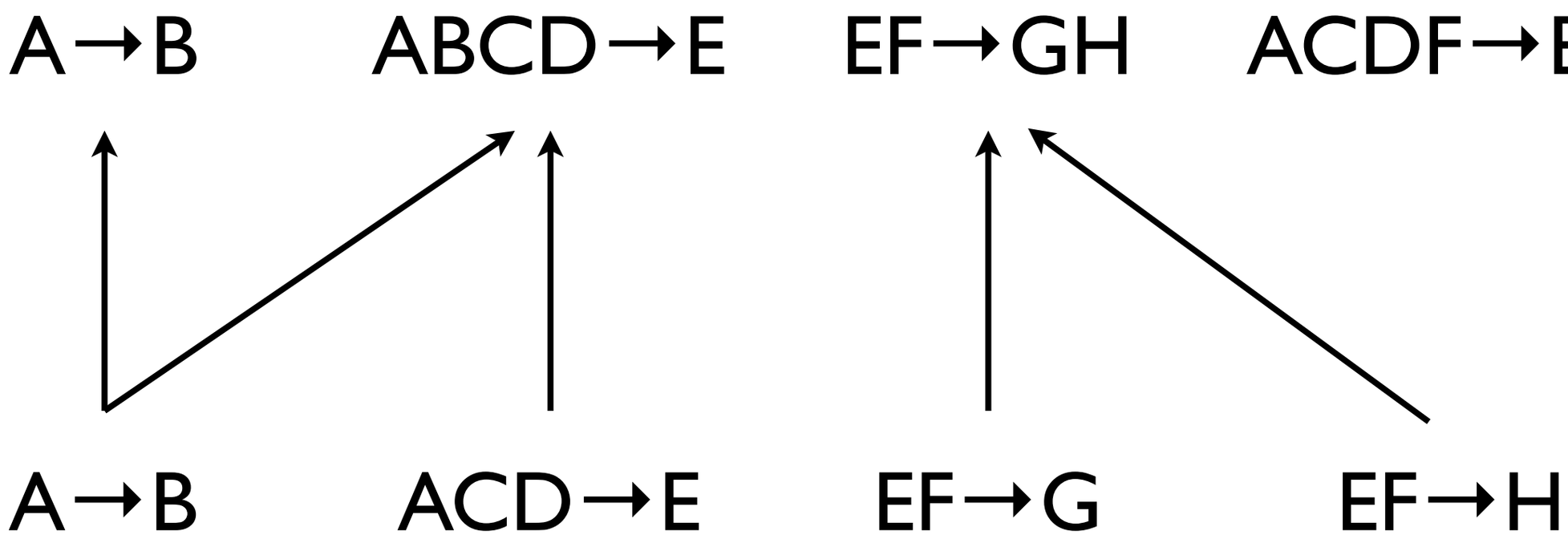


A→B  ACD→E  EF→G  EF→H

… is the minimal cover

# Example

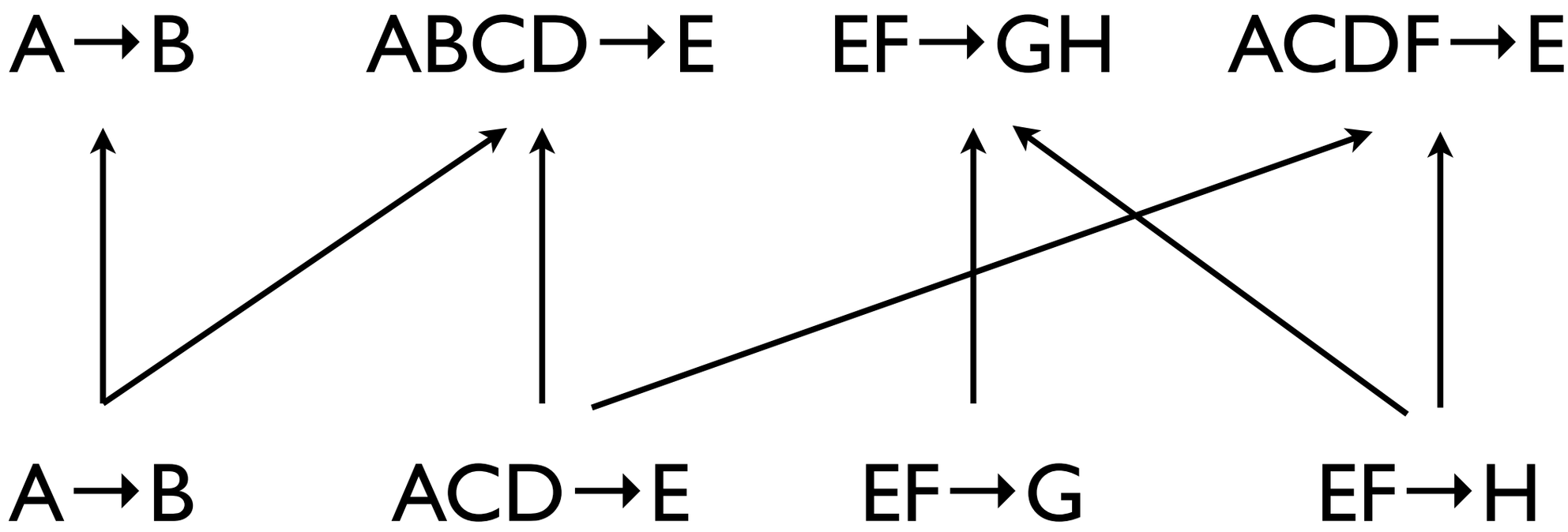
For the following set of FDs




… is the minimal cover

# Example

For the following set of FDs




… is the minimal cover

# Minimal Cover Algorithm

- **Standardize the FDs**: Split FDs using decomposition to make FDs with only one attribute on the RHS.
- **Minimize the FDs**: Remove attributes from the LHS of each FD until $F^+$ changes.
- **Delete redundant FDs**: Remove FDs until $F^+$ changes.

# Datalog

R&G Ch 24

# Recursion in SQL

| Officer | Direct Superior | Rank |
|---|---|---|
| Kirk | NULL | 4 |
| Spock | Kirk | 3 |
| McCoy | Kirk | 2.5 |
| Scotty | Kirk | 2.5 |
| Keller | Scotty | 1 |
| Sulu | Spock | 2 |
| Chapel | McCoy | 1 |
| Chekhov | Sulu | 1 |

# Recursion in SQL

| Officer | Direct Superior | Rank |
|---|---|---|
| Kirk | NULL | 4 |
| Spock | Kirk | 3 |
| McCoy | Kirk | 2.5 |
| Scotty | Kirk | 2.5 |
| Keller | Scotty | 1 |
| Sulu | Spock | 2 |
| Chapel | McCoy | 1 |
| Chekhov | Sulu | 1 |

Who can Spock give orders to?

# Recursion in SQL

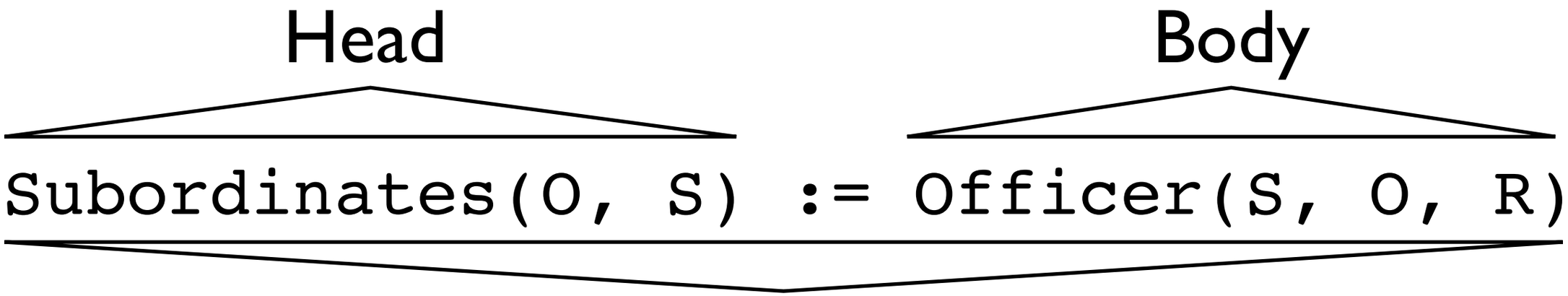
- SQL92 does not support recursion
  - Added in SQL99 via WITH RECURSIVE
  - View Defined via Transitive Closure
- Important class of queries require recursion.
- Simpler formulation of the same principles:
  - Datalog, a recursive query language

# Datalog

```
Subordinates(O, S) := Officer(S, O, R)
```

```
Subordinate(O, S) :=
    Officer(S, DS, R), Subordinate(DS,O)
```

# Datalog




```
Subordinate(O, S) :=
    Officer(S, DS, R), Subordinate(DS,O)
```

Atoms

# Datalog

If there is an officer ‘O’ with superior ‘S’,
then ‘S’ has a subordinate ‘O’

```
Subordinate(S, O) := Officer(O, S, R)
```

```
Subordinate(S, O) :=
    Officer(O, DS, R), Subordinate(DS,S)
```

If there is an officer ‘O’ with superior ‘DS’, and ‘DS’ is a subordinate of ‘S’
then ‘S’ has a subordinate ‘O’

# Datalog

If there is a tuple <O,S,R> in Officer,
then there is a tuple <S,O> in Subordinate

```
Subordinate(S, O) := Officer(O, S, R)
```

```
Subordinate(S, O) :=
    Officer(O, DS, R), Subordinate(DS,S)
```

If there is a tuple <O,DS,R> in Officer, and a tuple <DS,S> in Subordinate
then there is a tuple <S,O> in Subordinate

# Datalog

- Each rule is a template for making inferences, or deductions.
  - Datalog-based systems typically called ‘deductive databases’
- Equivalent in SQL: WITH RECURSIVE

# Recursive SQL

```
WITH RECURSIVE Subordinates(S, O) AS
   (SELECT Of.S,Of.O
    FROM Officer Of)
   UNION
   (SELECT Of.S,Su.O
    FROM Officer Of, Subordinate Su
    WHERE Of.O = Su.S)

SELECT O
FROM Subordinates
```

# Recursive SQL

```
WITH RECURSIVE Subordinates(S, O) AS
   (SELECT Of.S,Of.O
    FROM Officer Of)
   UNION
   (SELECT Of.S,Su.O
    FROM Officer Of, Subordinate Su
    WHERE Of.O = Su.S)

SELECT O
FROM Subordinates
WHERE S = 'Spock';
```

# Datalog

- How do we evaluate Datalog?
  - (How do we evaluate recursive SQL?)
- What does a Datalog program mean?
  - 1- Least Model Semantics
  - 2- Least Fixpoint Semantics

# Least Model

- If the Body is true, the Head must also be true.
  - Unidirectional, forces tuples into the head, but doesn't limit the size of the head:
  - Naive solution: Head consists of **all** tuples.
  - Better solution: We need some notion of a 'minimal solution' to the program.

# Least Model

- A model is a set of relation *instances* that satisfy a datalog program.
- For *every* constant valuation of *every* variable in *every* rule where *every* atom specifies a tuple that exists, the head also exists.
- A model can be bigger than necessary.
  - Least Model: A model contained in (smaller than) every other model for the program.

# Least Model

| Officer | Direct Superior | Rank |
|---|---|---|
| Kirk | NULL | 4 |
| Spock | Kirk | 3 |
| McCoy | Kirk | 2.5 |
| Scotty | Kirk | 2.5 |
| Keller | Scotty | 1 |
| Sulu | Spock | 2 |
| Chapel | McCoy | 1 |
| Chekhov | Sulu | 1 |

Can we add Subordinate(Scotty,Chekhov) to the result?

# Fixpoint

f(x)

# Fixpoint

f( f(x) )

# Fixpoint

f(f(f(x)))

# Fixpoint

f(…( f( f( f(x) ) ))…)

# Fixpoint

f(…( f( f( f(x) ) ))…)

A fixpoint of $f = f^n(x)$
such that $f^n(x) = f^{n+1}(x) = f^{n+2}(x) = \ldots$
(for some x)

# Fixpoint

Double(IntSet) =
IntSet ∪ (Every element of IntSet times 2)

Double({1,2,5}) = {1,2,5}∪{2,4,10} = {1,2,4,5,10}

What are some fixpoints of Double?

# Least Fixpoint

Need some basis for comparing function arguments
(e.g., Set containment)

Set of all even integers is smaller than Set of all integers

Least fixpoint is “smaller than” all other fixpoints.

May not exist: Does Double() have a least fixpoint?

# Fixpoint for Datalog

```
Subordinates(O, S) := Officer(S, O, R)

Subordinate(O, S) :=
    Officer(S, DS, R), Subordinate(DS,O)
```

# Fixpoint for Datalog

```
Subordinates(O, S) := Officer(S, O, R)

Subordinate(O, S) :=
    Officer(S, DS, R), Subordinate(DS,O)
```

Subordinate =

$\pi_{O,S}$(Officer ⋈ Subordinate) ∪ Subordinate

# Fixpoint for Datalog

```
Subordinates(O, S) := Officer(S, O, R)

Subordinate(O, S) :=
    Officer(S, DS, R), Subordinate(DS,O)
```

Subordinate =

$\pi_{O,S}$(Officer ⋈ Subordinate) ∪ Subordinate

Subordinate = f(Officer,Subordinate)

# Fixpoint for Datalog

```
Subordinates(O, S) := Officer(S, O, R)

Subordinate(O, S) :=
    Officer(S, DS, R), Subordinate(DS,O)
```

Subordinate =

$\pi_{O,S}$(Officer ⋈ Subordinate) ∪ Subordinate

Subordinate = f(Officer,Subordinate)

Subordinate = $f_{Officer}$(Subordinate)

# Fixpoint for Datalog

- Can compute Fixpoint by repeated application of the derivation function.
  - Apply function
    - Have new tuples been created?
      - If so, repeat.
      - Otherwise, done.
- Does this compute the least fixpoint?

# Adding Functionality

```
Subordinates(O, S) :=
        Officer(S, O, R), R ≥ 3
```

```
Subordinates(O, S, Opinion) :=
        Officer(S, O, R), R ≥ 3
```

# Safety

```
Subordinates(O, S, Opinion) :=
        Officer(S, O, R)
```

Where does ‘Opinion’ come from?

Subordinates now has an infinite number of rows.

This definition of Subordinates is unsafe.
Each variable must be range-restricted.

# Safety, Models, and Fixpoints

- A safe program has a finite least model.

- A safe (and negation free) program has a single least fixpoint identical to its least model.

# Negation

```
Ham(O) :=
    Officer(O,S,R), R = 4, NOT Stoic(O)

Stoic(O) :=
    Officer(O,S,R), NOT Ham(O)
```

http://tvtropes.org/pmwiki/pmwiki.php/Main/LargeHam
http://tvtropes.org/pmwiki/pmwiki.php/Main/TheStoic

# Negation

```
Ham(O) :=
    Officer(O,S,R), R = 4, NOT Stoic(O)

Stoic(O) :=
    Officer(O,S,R), NOT Ham(O)
```

Is Spock (Rank = 3) a ‘Large Ham’, or a ‘The Stoic’?

http://tvtropes.org/pmwiki/pmwiki.php/Main/LargeHam
http://tvtropes.org/pmwiki/pmwiki.php/Main/TheStoic

# Negation

```
Ham(O) :=
    Officer(O,S,R), R = 4, NOT Stoic(O)

Stoic(O) :=
    Officer(O,S,R), NOT Ham(O)
```

Is Spock (Rank = 3) a ‘Large Ham’, or a ‘The Stoic’?

Is Kirk (Rank = 4) a ‘Large Ham’, or a ‘The Stoic’?

http://tvtropes.org/pmwiki/pmwiki.php/Main/LargeHam
http://tvtropes.org/pmwiki/pmwiki.php/Main/TheStoic

# Negation

```
Ham(O) :=
    Officer(O,S,R), R = 4, NOT Stoic(O)

Stoic(O) :=
    Officer(O,S,R), NOT Ham(O)
```

Is Spock (Rank = 3) a ‘Large Ham’, or a ‘The Stoic’?

Is Kirk (Rank = 4) a ‘Large Ham’, or a ‘The Stoic’?

This program has TWO fixpoints!

http://tvtropes.org/pmwiki/pmwiki.php/Main/LargeHam
http://tvtropes.org/pmwiki/pmwiki.php/Main/TheStoic

# Stratification

- Table R depends on Table S if R is in the head of a rule with table S in its body.
  - R depends negatively on S if R is in the head of a rule with NOT S in its body.
- Classify tables into layers (or strata):
  - Stratum 0 has all tables with no dependencies
  - Stratum 1 depends on tables in S1, or depends negatively on tables in S0.
  - Stratum 2 depends on tables in S2 or negatively on S1…

# Stratification

- A Stratified Program is a program that can be organized into strata.

- Is LargeHam/TheStoic a Stratified Program?
- Does Stratification ensure a single fixpoint?
  - Why/Why not?