UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

# CS411 - Functional Dependencies

illinois.edu

# Announcements

- Project Deadlines
  - Stage 0 was due today
  - Stage 1 has been moved back to Feb 17th
  - Subsequent stages moved accordingly
- Tutorial session
  - Tuesday 2/12 at 6:00pm in 0216 SC
  - Bring a laptop

# MP Overview

- You'll be implementing relational algebra operators
- Each operator is a Python class that inherits from an abstract Operator superclass

# MP Overview

- In Python “self” refers to the current class instance
  - like “this” in C++ or Java
- “this.attribute” refers to an attribute
- “this.function()” refers to an instance method

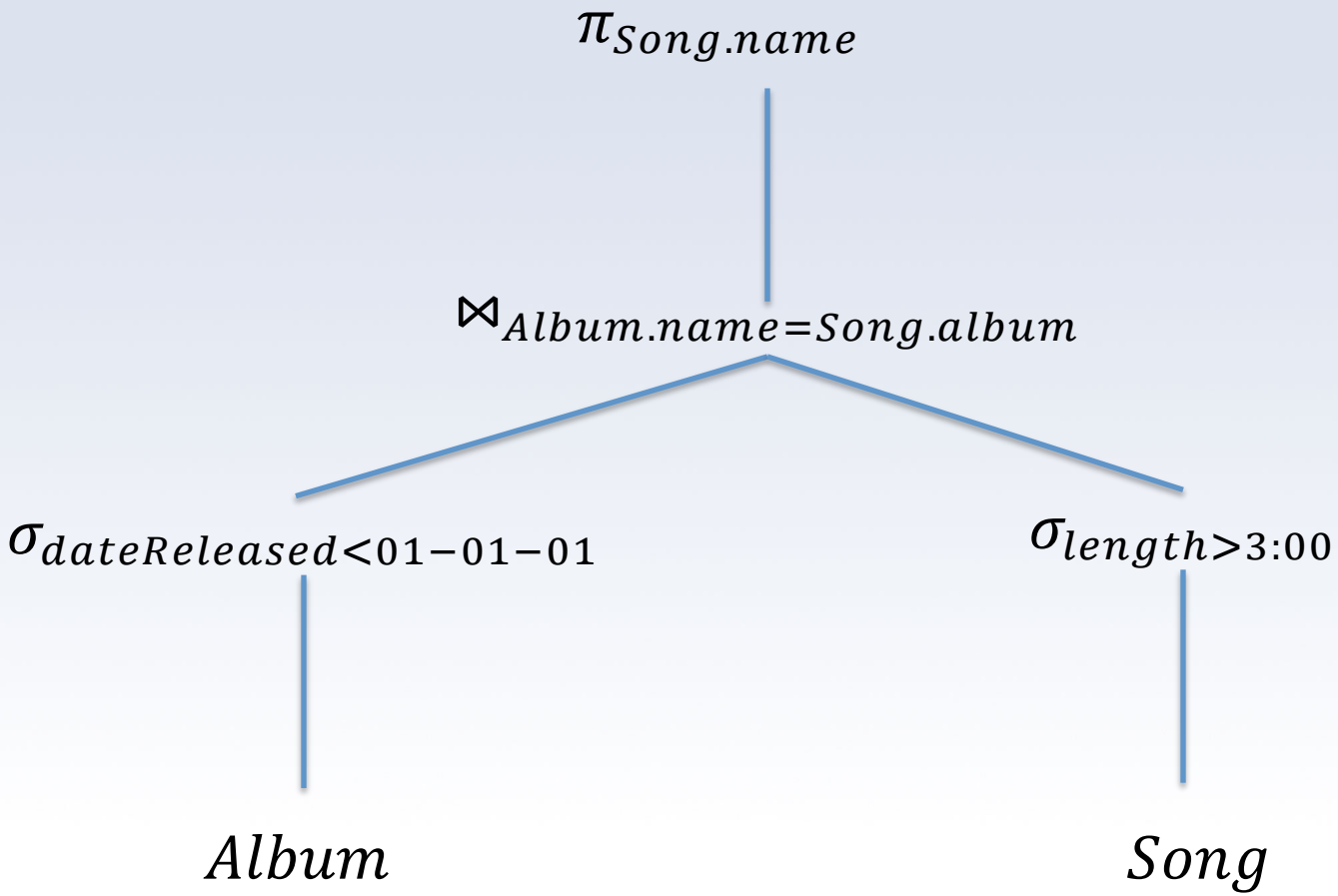
# MP Overview

- We implemented the constructors and build the query object for you
- You just implement two methods:
  - Check if the operation is valid
  - Perform the operation

# MP Overview

- Operators are nodes in a tree
  - Their children are also operations
- A relational algebra query is a "tree of subqueries"

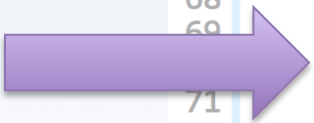
# MP Overview


$\pi_{Song.name}$
$\bowtie_{Album.name=Song.album}$
$\sigma_{dateReleased<01-01-01}$
$\sigma_{length>3:00}$
Album
Song

# MP Overview

- To determine if a operation is valid
  - check if children are valid
  - check if self is valid
- To perform the operation
  - perform the child operations
  - use the results from the children to perform our operation

# Run is_valid/do on children



```
53  def is_valid(self):
54      #Check if input relations are valid
55      (child_valid, child_name, child_attrs) = self.R.is_valid();
56      if not child_valid: return (False, None, None);
57      #Check if the input expression is valid
58      expr = self.expr;
59      for  attr in child_attrs:
60          expr = expr.replace(attr, "'"+attr+"'");
61      try:
62          eval(expr,dict());
63      except:
64          return (False, None, None);
65      return (True, child_name, child_attrs);
66
67  def do(self):
68      #Should return a Relation object representing the
69      #result of this operation being performed.'
70      self.R = self.R.do();
71      S = Relation(self.R.name, self.R.attributes);
72      expr = self._parseExpression(self.expr, S);
73      for t in self.R:
74          if eval(expr):
75              S.add(t);
76      return S;
```

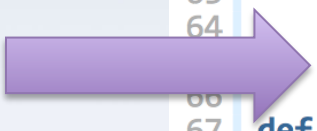
# MP Overview

- Part of checking if an operation is valid is checking the relation and attribute names
- is_valid() must return:
  - a boolean value indicating if we are valid
  - the name of our resulting relation
  - the list of attributes in our resulting the relation

# is_valid() returning a tuple

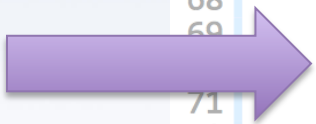
```
53  def is_valid(self):
54      #Check if input relations are valid
55      (child_valid, child_name, child_attrs) = self.R.is_valid();
56      if not child_valid: return (False, None, None);
57      #Check if the input expression is valid
58      expr = self.expr;
59      for  attr in child_attrs:
60          expr = expr.replace(attr, "'"+attr+"'");
61      try:
62          eval(expr,dict());
63      except:
64          return (False, None, None);
        return (True, child_name, child_attrs);

67  def do(self):
68      #Should return a Relation object representing the
69      #result of this operation being performed.'
70      self.R = self.R.do();
71      S = Relation(self.R.name, self.R.attributes);
72      expr = self._parseExpression(self.expr, S);
73      for t in self.R:
74          if eval(expr):
75              S.add(t);
76      return S;
```



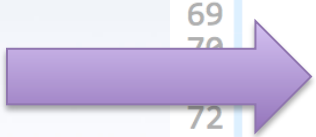
# 1. Execute Child Operators

```
53  def is_valid(self):
54      #Check if input relations are valid
55      (child_valid, child_name, child_attrs) = self.R.is_valid();
56      if not child_valid: return (False, None, None);
57      #Check if the input expression is valid
58      expr = self.expr;
59      for  attr in child_attrs:
60          expr = expr.replace(attr, "'"+attr+"'");
61      try:
62          eval(expr,dict());
63      except:
64          return (False, None, None);
65      return (True, child_name, child_attrs);
66
67  def do(self):
68      #Should return a Relation object representing the
69      #result of this operation being performed.'
70      self.R = self.R.do();
71      S = Relation(self.R.name, self.R.attributes);
72      expr = self._parseExpression(self.expr, S);
73      for t in self.R:
74          if eval(expr):
75              S.add(t);
76      return S;
```



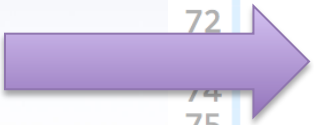
# 2. Create a new relation

```
53  def is_valid(self):
54      #Check if input relations are valid
55      (child_valid, child_name, child_attrs) = self.R.is_valid();
56      if not child_valid: return (False, None, None);
57      #Check if the input expression is valid
58      expr = self.expr;
59      for  attr in child_attrs:
60          expr = expr.replace(attr, "'"+attr+"'");
61      try:
62          eval(expr,dict());
63      except:
64          return (False, None, None);
65      return (True, child_name, child_attrs);
66
67  def do(self):
68      #Should return a Relation object representing the
69      #result of this operation being performed.'
70      self.R = self.R.do();
71      S = Relation(self.R.name, self.R.attributes);
72      expr = self._parseExpression(self.expr, S);
73      for t in self.R:
74          if eval(expr):
75              S.add(t);
76      return S;
```



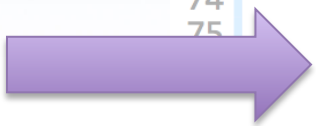
# 3. Populate the new relation

```
53  def is_valid(self):
54      #Check if input relations are valid
55      (child_valid, child_name, child_attrs) = self.R.is_valid();
56      if not child_valid: return (False, None, None);
57      #Check if the input expression is valid
58      expr = self.expr;
59      for  attr in child_attrs:
60          expr = expr.replace(attr, "'"+attr+"'");
61      try:
62          eval(expr,dict());
63      except:
64          return (False, None, None);
65      return (True, child_name, child_attrs);
66
67  def do(self):
68      #Should return a Relation object representing the
69      #result of this operation being performed.'
70      self.R = self.R.do();
71      S = Relation(self.R.name, self.R.attributes);
72      expr = self._parseExpression(self.expr, S);
73      for t in self.R:
74          if eval(expr):
75              S.add(t);
76      return S;
```



# 4. Return the finished relation

```
53  def is_valid(self):
54      #Check if input relations are valid
55      (child_valid, child_name, child_attrs) = self.R.is_valid();
56      if not child_valid: return (False, None, None);
57      #Check if the input expression is valid
58      expr = self.expr;
59      for  attr in child_attrs:
60          expr = expr.replace(attr, "'"+attr+"'");
61      try:
62          eval(expr,dict());
63      except:
64          return (False, None, None);
65      return (True, child_name, child_attrs);
66
67  def do(self):
68      #Should return a Relation object representing the
69      #result of this operation being performed.'
70      self.R = self.R.do();
71      S = Relation(self.R.name, self.R.attributes);
72      expr = self._parseExpression(self.expr, S);
73      for t in self.R:
74          if eval(expr):
75              S.add(t);
        return S;
```



# Review

- What SQL clauses did we learn?
  - Which one corresponds to selection?
  - Which one corresponds to projection?
- What other clauses did we learn?

# Review

- What does this query do?

```
SELECT COUNT(DISTINCT bandName)
FROM Musician NATURAL JOIN Band
WHERE lastName LIKE 'C%' AND
        age<>33;
```

# Big picture

- We now understand how to write queries against a database schema
- How do we design a database schema?
- How do we determine if it is “good”?

# Was this schema “good”?

Band(Name,City,Genre,YearFormed,YearEnded,Label)

Musician(FirstName,LastName,Band,Instrument)

Album(Title,Band,DateReleased)

Song(Title,Album,Length)

# Can we make it “better”?

Band(Name,City,Genre,YearFormed,YearEnded,Label)

Musician(FirstName,LastName,Band,Instrument)

Album(Title,Band,DateReleased)

Song(Title,Album,Length)

# Some general principles

- Given a database schema, can we transform it into a better schema?
  - preserve all the information in original
  - avoid redundancy
  - preserve dependencies and constraints
  - give good query performance

# Some general principles

- Given a database schema, can we transform it into a better schema?
  - To spot redundancies, we need a formal definition of *dependency*
  - Values are redundant if we can "figure them out" using other values
  - Storing age AND birthday is redundant

# Functional Dependency

- Defined for a particular relation R
- A set of its attributes $A_1,A_2,\ldots,A_n$ determine attributes $B_1,B_2,\ldots B_m$

# Functional Dependency

- If a tuple agrees on the *values* of $A_1, A_2, \ldots, A_n$
- Then they must agree on the *values* of $B_1, B_2, \ldots B_m$

# Functional Dependency

- $A_1$,$A_2$,…,$A_n$ “functionally determine” $B_1$,$B_2$, …$B_m$
- Written like:

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

# Example

Person(ssn,name,birthdate,address,age)

$$birthdate \rightarrow age$$

$$ssn \rightarrow name, birthdate, address, age$$

# Example

Class(courseNumber,roomNumber, instructorName,studentName,section, grade,TA)

$$section \rightarrow TA$$

$$student, courseNumber \rightarrow grade$$

$$courseNumber \rightarrow roomNumber, instructorName$$

# Vocabulary

- If, for every instance of relation R, a set of FDs F is true, then R *satisfies* F
  - Can't just be true for one instance
  - In this room, it's probably true that name→age
  - But it would not be true for the general population
  - In general, a Person relation doesn't satisfy F

# Keys

- We now have a way to formally define the notion of a key for a relation

# Keys

- A key is a set of attributes $\{A_1,A_2,\ldots,A_n\}$ for a relation R such that:
  1. $\{A_1,A_2,\ldots,A_n\}$ functionally determine all other attributes of R
  2. No subset of $\{A_1,A_2,\ldots,A_n\}$ functionally determines all other attributes of R

# Keys

1. $\{A_1,A_2,\ldots,A_n\}$ functionally determine all other attributes of R
   - If a tuple agrees on the key, they must agree on everything else
   - It is impossible for distinct tuples to agree on the key

# Keys

2. No subset of $A_1,A_2,\ldots,A_n$ functionally determines all other attributes of R
   - The key must be *minimal*

# Example

Person(ssn,name,birthdate,address,age)

$$ssn \rightarrow name, birthdate, address, age$$

- ssn is a key of the person relation
- what about {ssn,name}?

# Example

Class(courseNumber,roomNumber, instructorName,studentName,section, grade,TA)

$$section \rightarrow TA$$

$$student, courseNumber \rightarrow grade, section$$

$$courseNumber \rightarrow roomNumber, instructorName$$

# Superkeys

- A set of attributes that contains a key is called a *superkey*
  - "superset of a key"

# Example

Person(ssn,name,birthdate,address,age)

$$ssn \rightarrow name, birthdate, address, age$$

- ssn is a key of the person relation
- {ssn,name} is a superkey

# Some vocabulary

- Two sets of FDs S and T are *equivalent* if the set of relations satisfying S is the same as the set of relations satisfying T
- A set of FDs S *follows* from a set of FDs T if every relation that satisfies T satisfies S
- If S follows from T and T follows from S, then S and T are equivalent

# Reasoning about FDs

- Given a relation and a set of functional dependencies
  - can we discover new functional dependencies?
  - can we discover all functional dependencies?
- Seems like we need a set of rules for manipulating FDs

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

Does AB → GH follow from F?

# Armstrong's Axioms

1. Reflexivity
2. Augmentation
3. Transitivity

# Armstrong's Axioms

1. Reflexivity ("trivial FDs")

$$B_1, B_2, \ldots, B_m \subseteq A_1, A_2, \ldots, A_n \Longrightarrow$$

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

2. Augmentation
3. Transitivity

# Armstrong's Axioms

1. Reflexivity
2. Augmentation

$$A_1, A_2, \dots, A_n \rightarrow B_1, B_2, \dots, B_m \implies$$

$$A_1, A_2, \dots, A_n, C_1, C_2, \dots, C_k \rightarrow B_1, B_2, \dots, B_m, C_1, C_2, \dots, C_k$$

$$\forall C_1, C_2, \dots, C_k$$

3. Transitivity

# Armstrong's Axioms

1. Reflexivity
2. Augmentation
3. Transitivity

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

$$B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_k$$

$$\Longrightarrow A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_k$$

# More rules

4. Union

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$
$$A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_k \Longrightarrow$$
$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m, C_1, C_2, \ldots, C_k$$

5. Decomposition
6. Pseudotransitivity

# More rules

4. Union
5. Decomposition

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m, C_1, C_2, \ldots, C_k$$

$$\Longrightarrow A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

$$A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_k$$

6. Pseudotransitivity

# More rules

4. Union
5. Decomposition
6. Pseudotransitivity

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

$$B_1, B_2, \ldots, B_m, C_1, C_2, \ldots, C_k \rightarrow D_1, D_2, \ldots, D_j$$

$$\Longrightarrow$$

$$A_1, A_2, \ldots, A_n, C_1, C_2, \ldots, C_k \rightarrow D_1, D_2, \ldots, D_j$$

# All the rules

1. Reflexivity
2. Augmentation
3. Transitivity
4. Union
5. Decomposition
6. Pseudotransitivity

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

Does AB → GH?

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

| | |
|---|---|
| AB→ E | (Given) |
| AB→AB | (1. Reflexivity) |
| AB→B | (5. Decomposition) |
| AB→BE | (4. Union) |
| BE→I | (Given) |

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

| | |
|---|---|
| AB → E | (0. Given) |
| AB → AB | (1. Reflexivity) |
| AB → B | (5. Decomposition) |
| AB → BE | (4. Union) |
| BE → I | (0. Given) |

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

| | |
|---|---|
| AB → I | ( 3. Transitivity) |
| E → G | (0. Given) |
| AB → G | (3. Transitivity) |
| AB → GI | (4. Union) |
| GI → H | (0. Given) |

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

| | |
|---|---|
| AB → H | (3. Transitivity) |
| GI → GI | (1. Reflexivity) |
| GI → I | (5. Decomposition) |
| AB → GH | (4. Union) |

# Attribute Closure

- Given a set of attributes $\{A_1,A_2,\ldots,A_n\}$ and S is a set of FDs
- Find all attributes B such that $\{A_1,A_2,\ldots,A_n\} \rightarrow B$
- We call the set of all such B's the *closure* of $\{A_1,A_2,\ldots,A_n\}$
- We use the notation $\{A_1,A_2,\ldots,A_n\}^+$

# Attribute Closure Algorithm

- INPUT:
  - a set of attributes $\{A_1,A_2,\ldots,A_n\}$
  - a set of FDs S
- OUTPUT:
  - The closure $\{A_1,A_2,\ldots,A_n\}^+$

# Attribute Closure Algorithm

1. Use decomposition (Rule 5) to split all FDs in S so the right hand side has one attribute
2. Initialize $X=\{A_1,A_2,\ldots,A_n\}$
3. Loop
   1. Find C such that C is not in X, $B_1,B_2,\ldots,B_m$ are in X, and $B_1,B_2,\ldots,B_m \rightarrow C$
   2. If C exists, add it to X. If not, break.
4. Return X

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→AD,D→E,CF→B}

Find $\{A,B\}^+$

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→AD,D→E,CF→B}

Find $\{A,B\}^+$

1. Split up F using decomposition:
   F={AB→C,BC→A, BC→D,D→E,CF→B}

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→AD,D→E,CF→B}

Find $\{A,B\}^+$

1. Split up F using decomposition:
   F={AB→C,BC→A, BC→D,D→E,CF→B}
2. Initialize X={AB}

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→A, BC→D,D→E,CF→B}

Find $\{A,B\}^+$

3. X={A,B}

Find new attributes to add to X

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→A, BC→D,D→E,CF→B}

Find $\{A,B\}^+$

3. X={A,B,C}

Find new attribute to add to X

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→A, BC→D,D→E,CF→B}

Find $\{A,B\}^+$

3. X={A,B,C,D}

Find new attribute to add to X

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→A, BC→D,D→E,CF→B}

Find $\{A,B\}^+$

3. X={A,B,C,D,E}

Find new attribute to add to X

# Example

R(A,B,C,D,E,F)

F={AB→C,BC→A, BC→D,D→E,CF→B}

Find $\{A,B\}^+$

3. X={A,B,C,D,E}
Nothing new to add, so

4. Return X={A,B,C,D,E}

# Uses of attribute closure

- Text if X is a superkey
  - Check if all attributes are in $X^+$
- Check if some FD (e.g. A→B) holds
  - Check if Y is in $X^+$
  - Easier than applying all the rules!

# Example

R(A,B,C,D,E,F,G,H,I,J)

F={AB → E, AG → J, BE → I, E → G, GI → H}

Does AB → GH?

$\{A,B\}^+ = \{A,B,E,I,G,H,J\}$

# Reminder

- Two sets of FDs S and T are *equivalent* if the set of relations satisfying S is the same as the set of relations satisfying T

# More vocabulary

- If we are given S, all equivalent sets of FDs T is called a *basis* for S
- A *minimal basis* satisfies
  1. All the FDs have singleton right sides
  2. If any FD is removed, it is no longer a basis
  3. If we remove one or more attributes from the left side of a FD, it is no longer a basis

# Example

R(A,B,C)

F={A→ B,A→ C,B→ A,B→ C,C→ A,C→B}

$B_1$={A→ B, B→ A, B→ C, C→ B}

$B_2$={A→ B, B→ C, C→ A}

# Projecting FDs

- Say we take the projection of a relation R with FDs F
- $S=\pi_L(R)$
- What FDs hold for S?
  - For each subset $A_i$ of L, compute $A_i^+$
  - Throw out attributes not in S
  - Union them all together

# Example

R(A,B,C,D)

F={A→B,B→C,C→D}

$S=\pi_{A,C,D}(R)$

Find G (FDs for S)

# Example

$R(A,B,C,D)$

$F=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

$S=\pi_{A,C,D}(R)$

$\{A\}^{+}=\{A,B,C,D\}$

$G=\{A \rightarrow C,\ A \rightarrow D\}$

# Example

$R(A,B,C,D)$

$F=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

$S=\pi_{A,C,D}(R)$

$\{C\}^{+}=\{C,D\}$

$G=\{A \rightarrow C,\ A \rightarrow D,\ C \rightarrow D\}$

# Example

$R(A,B,C,D)$

$F=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

$S=\pi_{A,C,D}(R)$

$\{D\}^{+}=\{D\}$

$G=\{A \rightarrow C,\ A \rightarrow D,\ C \rightarrow D\}$

# Example

$R(A,B,C,D)$

$F=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

$S=\pi_{A,C,D}(R)$

$\{A,C\}^{+}=\{A,B,C,D\}$

$G=\{A \rightarrow C,\ A \rightarrow D,\ C \rightarrow D\}$

# Example

R(A,B,C,D)

F={A→B,B→C,C→D}

$S=\pi_{A,C,D}(R)$

Skipping some uninteresting steps…

# Example

$R(A,B,C,D)$

$F=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

$S=\pi_{A,C,D}(R)$

$\{C,D\}^{+}=\{D\}$

$G=\{A \rightarrow C,\ A \rightarrow D,\ C \rightarrow D\}$