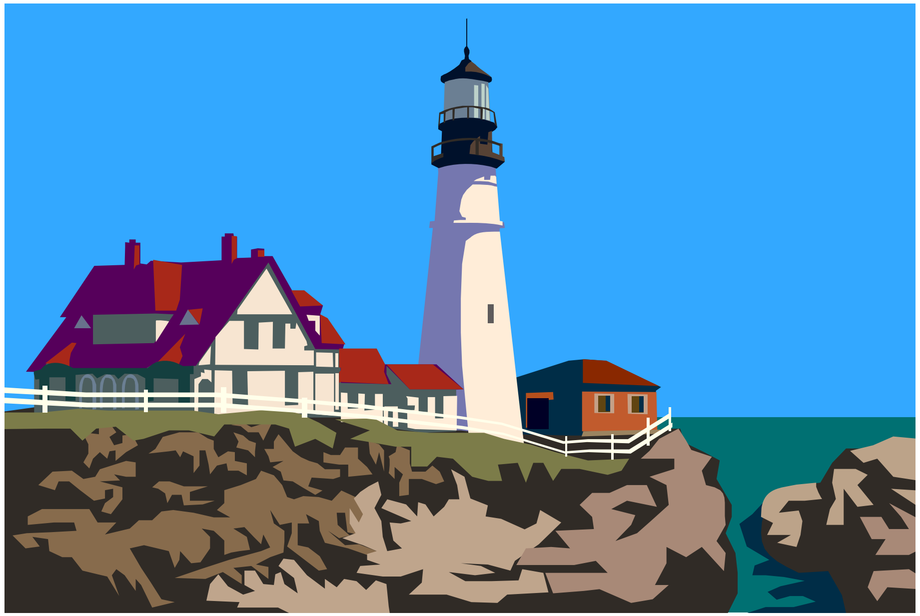
Detecting Functional Dependencies

21.5.2013

Felix Naumann

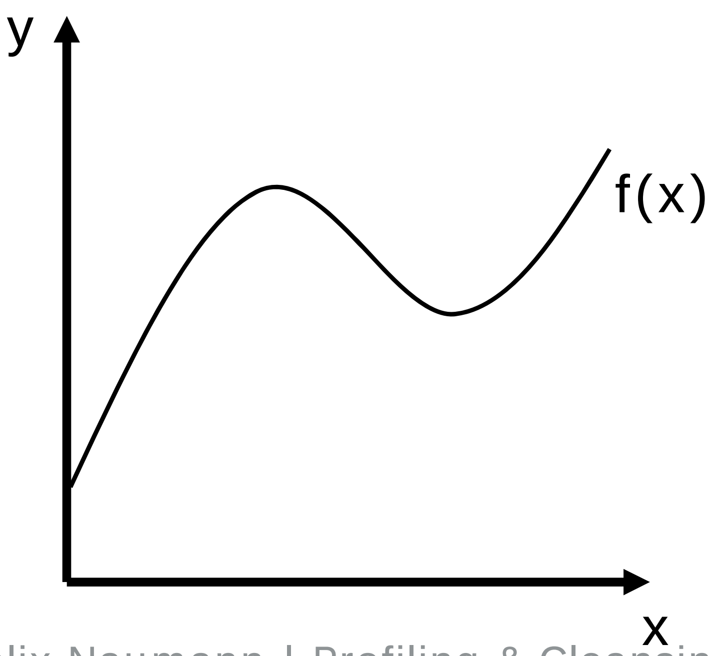
Overview

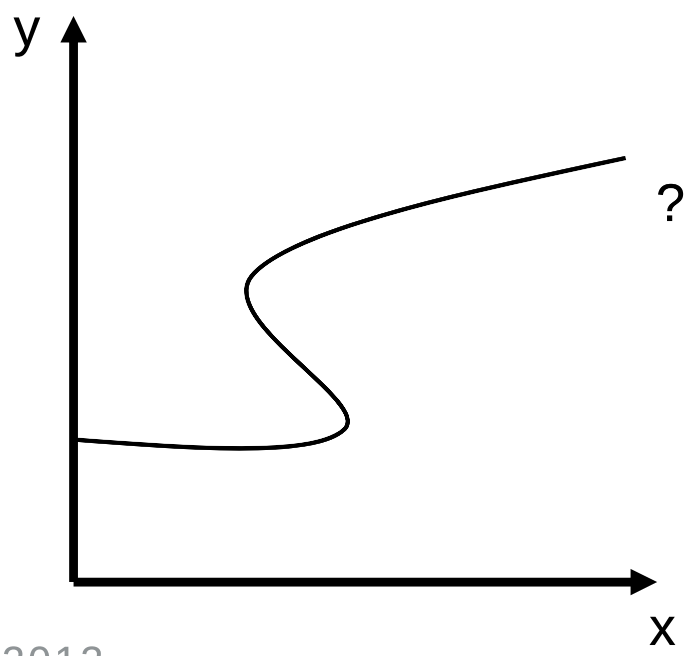
- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



Definition – Functional Dependency

- „$X \rightarrow A$“ is a statement about a relation R: When two tuples have same value in attribute set X, the must have same values in attribute A.
- Formally: $X \rightarrow A$ is an FD over R $(R \vDash X \rightarrow A) \Leftrightarrow$ for all tuples $t_1, t_2 \in R$: $t_1[X] = t_2[X] \Rightarrow t_1[A] = t_2[A]$
- Can generalize to sets: $X \rightarrow Y \Leftrightarrow X \rightarrow A$ for each $A \in Y$





Trivial FDs

- Trivial: Attributes on RHS are subset of attributes on LHS
  - Street, City → City
  - Any trivial FD holds
- Non-trivial: At least one attribute on RHS does not appear on LHS
  - Street, City → Zip, City
- Completely non-trivial: Attributes on LHS and RHS are disjoint.
  - Street, City → Zip
- Minimal FD: RHS does not depend on any subset of LHS.
- Typical goal: Given a relation R, find all minimal completely non-trivial functional dependencies.

FD Inference Rules

- R1 Reflexivity $X \supseteq Y \Rightarrow X \rightarrow Y$ (also $X \rightarrow X$)
  - Trivial FDs
- R2 Accumulation $\{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ$
  - Aka: Augmentation
- R3 Transitivity $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z$

- R1-R3 known as *Armstrong-Axioms*
  - Sound and complete

- R4 Decomposition $\{X \rightarrow YZ\} \Rightarrow X \rightarrow Y$
- R5 Union $\{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ$
- R6 Pseudotransitivity $\{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z$

FD Discussion

- Schema vs. instance
- Keys as special case for FDs
  - X is key of R if X → R\X

- Uses for FDs
  - Schema design and normalization
  - Key discovery
  - Data cleansing (especially conditional FDs)

Naive Discovery Approach

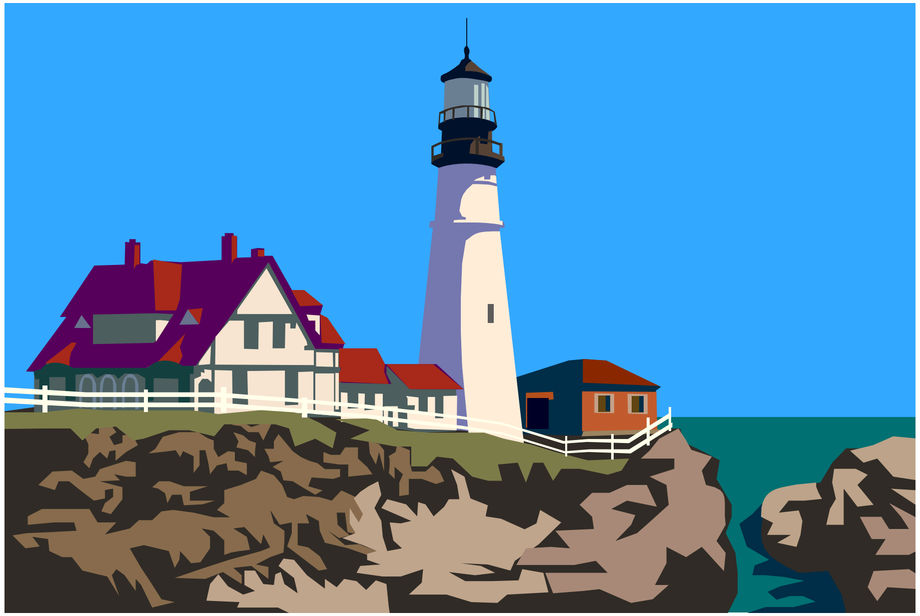
- Given relation R, detect all minimal, non-trivial FDs X → A.

- For each column combination X
  - For each pair of tuples (t1,t2)
    - If t1[X\A] = t2[X\A] and t1[A] ≠ t2[A]: Break

- Complexity
  - Exponential in number of attributes
  - times number of rows squared

Overview

- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



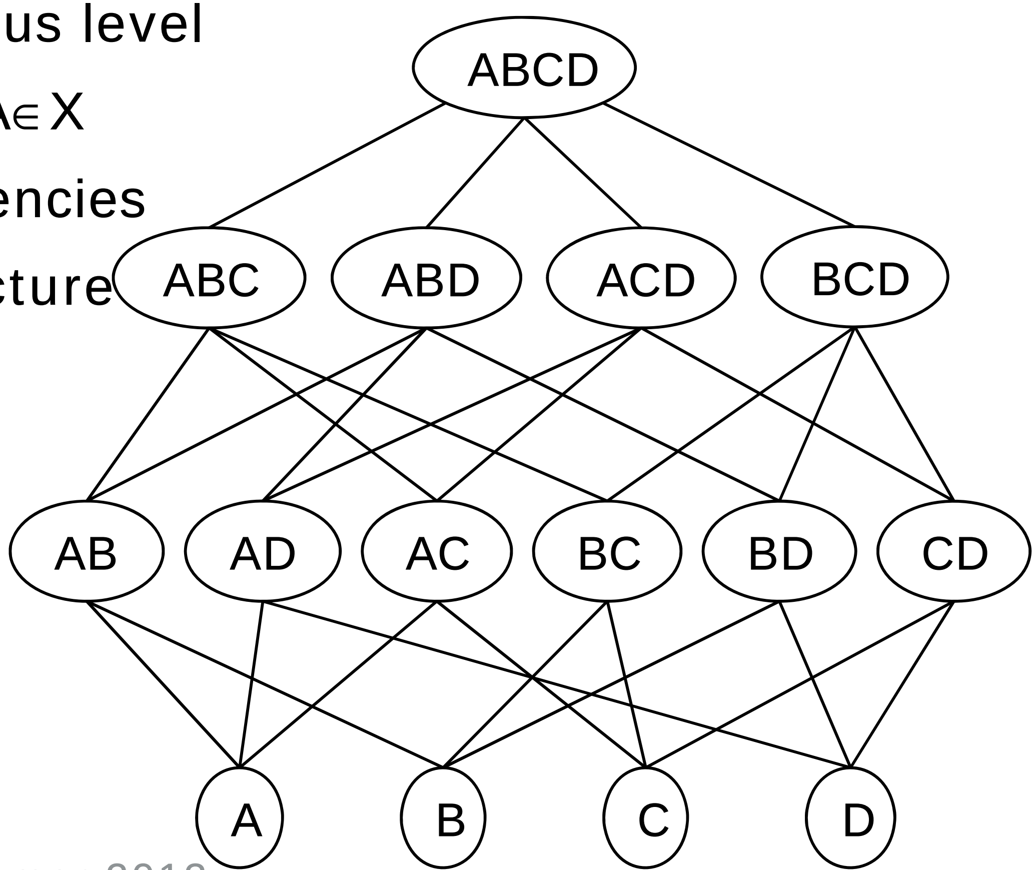
Tane – General Idea

- Two elements of approach
  1. Reduce column combinations through pruning
     - Reasoning over FDs
  2. Reduce tuple sets through partitioning
     - Partition data according to attribute values
     - Level-wise increase of size of attribute set
       - Consider sets of tuples whose values agree on that set

- Huhtala, Y.; Kärkkäinen, J.; Porkka, P. & Toivonen, H.
  TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies
  Computer Journal, **1999**, *42*, 100-111

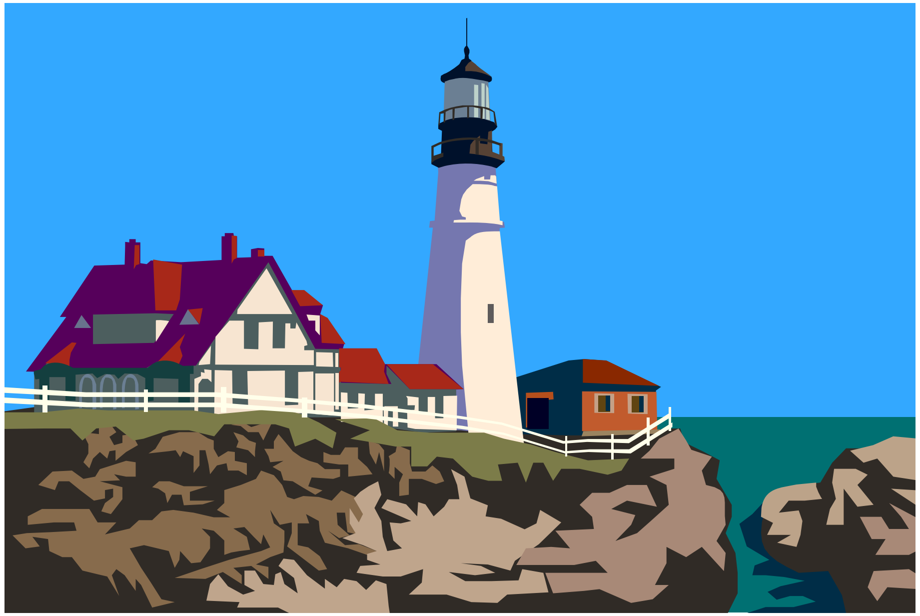
Discovery strategy

- Bottom up traversal through lattice
  - $\Rightarrow$ only minimal dependencies
  - Pruning
  - Re-use results from previous level
- For a set X, test all $X \setminus A \rightarrow A$, $A \in X$
  - $\Rightarrow$ only non-trivial dependencies
  - Test on efficient data structure



Overview

- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



Candidate Sets

- RHS candidate set C(X)
- Stores only those attributes that might depend on **all** other attributes of X.
  - I.e., those that still need to be checked
  - If $A \in C(X)$ then A does not depend on any proper subset of X.
- $C(X) = R \setminus \{A \in X \mid X \setminus A \rightarrow A \text{ holds}\}$

- Example: R= {ABCD}, and $A \rightarrow C$ and $CD \rightarrow B$
  - $C(A) = \{ABCD\} \setminus \{\} = C(B) = C(C) = C(D)$
  - $C(AB) = \{ABCD\} \setminus \{\}$
  - $C(AC) = \{ABCD\} \setminus \{C\} = \{ABD\}$
  - $C(CD) = \{ABCD\} \setminus \{\}$
  - $C(BCD) = \{ABCD\} \setminus \{B\} = \{ACD\}$

RHS candidate pruning

- For minimality it suffices to test $X \setminus A \rightarrow A$ where
  - $A \in X$ and $A \in C(X \setminus \{B\})$ for all $B \in X$.
  - I.e., A is in **all** candidate sets of the subsets.

- Example
  - $X = \{ABC\}$. Assume we know $C \rightarrow A$ from previous step.
  - Need to test three dependencies: $AB \rightarrow C$, $AC \rightarrow B$, and $BC \rightarrow A$
    - We should not be testing $BC \rightarrow A$, because we know $C \rightarrow A$
  - Candidate sets:
    - $C(AB) = \{ABC\}$, $C(AC) = \{BC\}$, $C(BC) = \{ABC\}$
  - E.g. $BC \rightarrow A$ does not need to be tested for minimality, because A is not in all three candidate sets: $A \notin C(AB) \cap C(AC) \cap C(BC)$
  - $AB \rightarrow C$, $AC \rightarrow B$ need to be tested, because B and C appear in all candidate sets.

Improved RHS candidate pruning

- Basis: Let $B \in X$ and let $X \setminus B \rightarrow B$ hold. If $X \rightarrow A$, then $X \setminus B \rightarrow A$.
  - Example: $A \rightarrow B$ holds. If $AB \rightarrow C$ holds, then also $A \rightarrow C$.
  - Use this to reduce candidate set: If $X \setminus B \rightarrow B$ for some B, then any dependency with X on LHS cannot be minimal.
    - Just remove B.

- $C^+(X) = \{A \in R \mid \forall B \in X: X \setminus \{A,B\} \rightarrow B$ does not hold$\}$
  - Special case: A = B corresponds to C(X)
  - $C(X) = R \setminus \{A \in X \mid X \setminus A \rightarrow A$ holds$\}$

- This definition removes three types of candidates.
  - $C1 = \{A \in X \mid X \setminus A \rightarrow A$ holds$\}$ (as before)
  - $C2 = \{R \setminus X\}$ if $\exists B \in X: X \setminus B \rightarrow B$
  - $C3 = \{A \in X \mid \exists B \in X \setminus A : X \setminus \{A,B\} \rightarrow B$ holds$\}$

Example for C2

- $C^{+}(X) = \{A \in R \mid \forall B \in X: X \setminus \{A,B\} \rightarrow B \text{ does not hold}\}$
- $C2 = \{R \setminus X\}$ if $\exists B \in X: X \setminus B \rightarrow B$

- R = ABCD, X = ABC
- C(X) = ABCD initially
- Discovery of C→B
  - Remove B from C(X)
  - Additionally remove R\X = D
  - Ok, because remaining combination of LHS contains B and C.
    - ABC→D is not minimal because C→B
- Together: $C^{+}(X) = \{AC\}$

C3

- $C3 = \{A \in X \mid \exists B \in X \setminus A : X \setminus \{A,B\} \rightarrow B \text{ holds}\}$
  - Same idea as before, but for subsets
- Assume X has proper subset Y ($X \supset Y$) such that $Y \setminus B \rightarrow B$ holds for some $B \in Y$.
- Then we can remove from C(X) all $A \in X \setminus Y$.

- Example X = ABCD and let $C \rightarrow B$
- $X \supset Y = BC$ and $X \setminus Y = AD$
- Thus can remove all AD.
  - Any remaining combination of LHS contains B and C.
    - $ABC \rightarrow D$ and $BCD \rightarrow A$
  - Again, since $C \rightarrow B$ any such FD is not minimal.
- Together: $C^+(X) = \{C\}$

More pruning of lattice: Key pruning

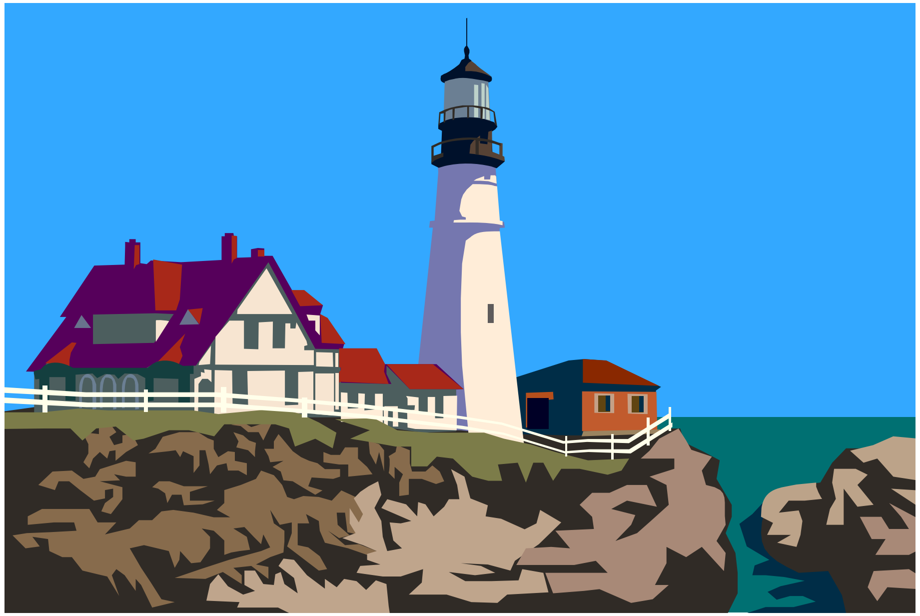
- Insight: If X is superkey and X\B →B, then X\B is also a superkey.

- Case 1: If X is superkey, no need to test any X → A.
- Case 2:
  - If X is superkey and not key, any X → A is not minimal (for any A∉X).
  - If A∈X and X\A→A then X\A is superkey, and no need to test.

- Summary: Can prune all keys and their supersets
- Later: Test for superkey-property based on “key-error” of partition

Overview

- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



TANE Base Algorithm

$L_0 := \{\emptyset\}$
$\mathcal{C}^+(\emptyset) := R$
$L_1 := \{\{A\} \mid A \in R\}$
$\ell := 1$
while $L_\ell \neq \emptyset$
  COMPUTE_DEPENDENCIES(L_ℓ)
  PRUNE(L_ℓ)
  $L_{\ell+1}$:= GENERATE_NEXT_LEVEL(L_ℓ)
  $\ell := \ell + 1$

- Each level L contains the corresponding nodes of the lattice

Generating Lattice Levels

- $L_{l+1} = \{X \mid |X| = l+1$ and all subsets $Y \subset X$ of size l are in $L_l\}$
  - General apriori idea
  - Can use L_l to generate L_{l+1}

1 $L_{\ell+1} := \emptyset$
2 **for each** $K \in$ PREFIX_BLOCKS(L_ℓ) **do**
3   **for each** $\{Y, Z\} \subseteq K, Y \neq Z$ **do**
4         $X := Y \cup Z$
5   **if** for all $A \in X, X \setminus \{A\} \in L_\ell$ **then**
6             $L_{\ell+1} := L_{\ell+1} \cup \{X\}$
7 **return** $L_{\ell+1}$

- Prefix blocks: disjoint sets from L_l with common prefix of size $l-1$
  - All pairs for $l = 1$
- Line 5: All subsets of new set must appear in lower level

Dependency Computation

HPI Hasso Plattner Institut

```
1  for each X ∈ L_ℓ do
2      C+(X) := ⋂_{A∈X} C+(X \ {A})
3  for each X ∈ L_ℓ do
4      for each A ∈ X ∩ C+(X) do
5          if X \ {A} → A is valid then
6              output X \ {A} → A
7              remove A from C+(X)
8              remove all B in R \ X from C+(X)
```

Trivial for L1:
Nothing happens

- Line 2: Create candidate sets; each attribute must appear in all candidate sets of smaller size
- Line 4: Only test attributes from candidate set
- Line 5: Actual test on data
- Line 7: Reduce candidates by newly found dependent
- Line 8: Reduce candidates by all other attributes: cannot depend on all others, because any combination involving A and LHS is not minimal

Pruning

1 **for each** $X \in L_\ell$ **do**
2   **if** $\mathcal{C}^+(X) = \emptyset$ **do**
3   delete X from L_ℓ
4   **if** X is a (super)key **do**
5   **for each** $A \in \mathcal{C}^+(X) \setminus X$ **do**
6   **if** $A \in \bigcap_{B \in X} \mathcal{C}^+(X \cup \{A\} \setminus \{B\})$ **then**
7   output $X \rightarrow A$
8   delete X from L_ℓ

- Line 3: Basic pruning
  - Deletion from L_l ensures that supersets cannot be created during level generation (loops not executed on empty candidate sets)
- Lines 4-8: Key pruning

Example run

- R = ABCD, $C \rightarrow B$ and $AB \rightarrow D$ are to be discovered
  - Also: $AC \rightarrow D$ through pseudo-transitivity
- $L_0 = \{\}$,
  - $C^+(\{\}) = ABCD$
  - nothing to do
- $L_1 = \{A\}, \{B\}, \{C\}, \{D\}$
  - $C^+(X) = ABCD$ for all $X \in L_1$
  - Still nothing to do: No FDs can be generated from singletons
  - Thus, no pruning
- L_2 = AB, AC, AD, BC, BD, CD
  - E.g. $C^+(AB) = C^+(AB \setminus A) \cap C^+(AB \setminus B) = ABCD \cap ABCD$
  - $C^+(X) = ABCD$ for all $X \in L_2$
  - Dep. checks for AB: $A \rightarrow B$ and $B \rightarrow A$ Nothing happens

Example run

- L_2 = AB, AC, AD, BC, BD, CD
  - $C^+(X)$ = ABCD for all $X \in L_2$
  - Dep. checks for BC: B→C (no!) and C→B (yes!)
  - Output C→B
  - Delete B from $C^+(BC)$: ACD
  - Delete R\BC from $C^+(BC)$: C
    - Note BC→A and BC→D are not minimal
- L_3 = ABC, ABD, ACD, BCD
  - $C^+(ABC) = C^+(AB) \cap C^+(AC) \cap C^+(BC)$ = C
  - $C^+(BCD) = C^+(BC) \cap C^+(BD) \cap C^+(CD)$ = C
  - $C^+(ABD) = C^+(ACD)$ = ABCD unchanged
  - Dep. check for ABC: ABC $\cap$ $C^+(ABC)$ are candidates
    - AB→C no! Did not check BC→A and AC→B

Example run

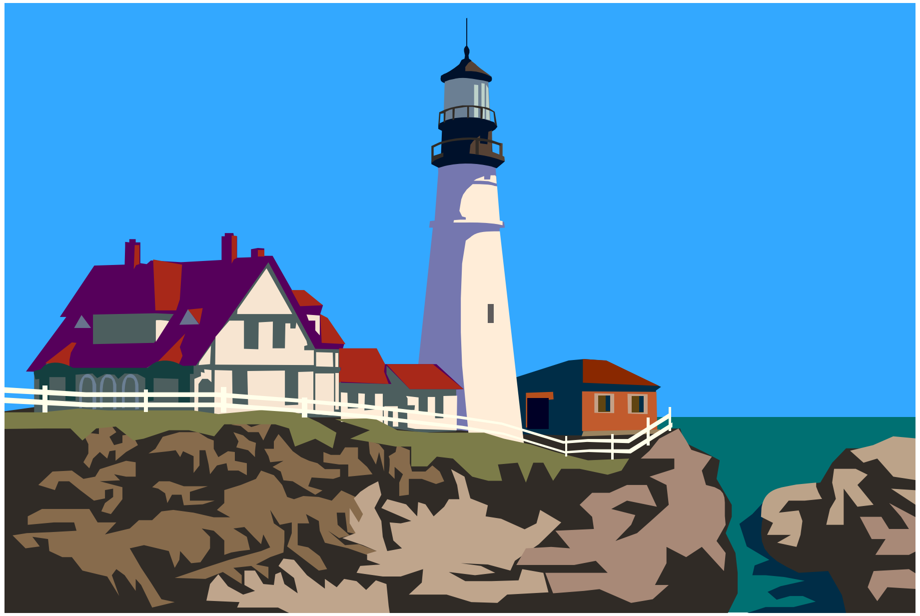
- L_3 = ABC, ABD, ACD, BCD
  - C^+(ABC) = C^+(BCD) = C
  - C^+(ABD) = C^+(ACD) = ABCD
  - Dep. check for ABD: ABD ∩ C^+(ABD) are candidates
    - AD→B and BD→A: no!
    - AB→D: yes! Output AB→D
    - Delete D from C^+(ABD): ABC
    - Delete R\ABD from C^+(ABD): AB
  - Dep. check for BCD: BCD ∩ C^+(BCD) are candidates
    - Only need to check BD→C: no!
  - Dep. check for ACD: ACD ∩ C^+(ACD) are candidates
    - CD→A and AD→C: no!
    - AC→D: yes! Output AC→D
    - Delete D from C^+(ABD): ABC
    - Delete R\ACD from C^+(ABD): AC

Example run

- L_4 = ABCD
  - $C^+(ABCD) = C^+(ABC) \cap C^+(ABD) \cap C^+(ACD) \cap C^+(BCD) = \{\}$
  - Nothing to check
  - Did not need to check
    - BCD→A: Not minimal because C → B
    - ACD→B: Not minimal because C → B
    - ABD→C: Not minimal because AB → D
    - ABC→D: Not minimal because AB → D

Overview

- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



Notation and Partitions

- Tuples t and u are equivalent wrt. attribute set X if t[A] = u[A] for all $A \in X$.

- Attribute set X partitions R into equivalence classes
  - Equivalence class of tuple t:

    $$[t]_X = \{u \in R \mid \forall A \in X : t[A] = u[A]\}$$

  - Partition is set of disjoint sets: $\pi_X = \{[t]_X \mid t \in R\}$
    - Each set has unique values for X-values.
  - $|\pi|$ is number of equivalence classes in π.

Partitioning - Example

TupleID	A	B	C	D
1	1	a	$	Flower
2	1	A		Tulip
3	2	A	$	Daffodil
4	2	A	$	Flower
5	2	b		Lily
6	3	b	$	Orchid
7	3	C		Flower
8	3	C	#	Rose

- $[1]_A = [2]_A = \{1,2\}$
- $\pi_A = \{\{1,2\}, \{3,4,5\}, \{6,7,8\}\}$
- $\pi_{BC} = \{\{1\}, \{2\}, \{3,4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$
- $\pi_D = \{\{147\}, \{2\}, \{3\}, \{5\}, \{6\}, \{8\}\}$

Partition refinement

- Partition π *refines* partition π' if every equivalence class in π is a subset of some equivalence class in π'.
  - π has a finer partitioning than π'.
- $X \rightarrow A \Leftrightarrow \pi_X$ refines π_A
  - If π_X refines π_A then $\pi_{X \cup A} = \pi_A$.
  - $\pi_{X \cup A}$ always refines π_A.
  - $\Rightarrow$ if $\pi_{X \cup A} \neq \pi_A$ then $|\pi_X| \neq |\pi_{X \cup A}|$.
  - $\Rightarrow$ if $|\pi_X| = |\pi_{X \cup A}|$ then $\pi_{X \cup A} = \pi_A$.
- Together: $X \rightarrow A \Leftrightarrow \pi_X$ refines $\pi_A \Leftrightarrow |\pi_X| = |\pi_{X \cup A}|$
  - This implies a simple check for an FD.
- π_A = {12, 345, 678}
- π_B = {12345, 678}
- π_{AB} = {12, 345, 678}

	A	B
1	1	A
2	1	A
3	2	A
4	2	A
5	2	A
6	3	B
7	3	B
8	3	B

Stripped partitions

- Idea: Remove equivalence classes of size 1 from partitions.
  - Singleton equivalence class cannot violate any FD.
  - Same idea as for position lists
- Problem
  - $X \rightarrow A \Leftrightarrow |\pi_X| = |\pi_{X \cup A}|$ not true for stripped partitions π'
  - $\Rightarrow$: $|\pi_C| = |\pi_{C \cup A}| = 6$ and $|\pi'_C| = |\pi'_{C \cup A}| = 2$
  - $\Leftarrow$: $|\pi'_B| = |\pi'_{B \cup C}| = 2$ but $B \nrightarrow C$
- Solution: Key error
  - $e(X)$ is minimum fraction of tuples to remove for X to be key
  - $e(X) = 1 - |\pi_X|/r$
    - $e(B) = 1 - |\pi_B|/r = 1 - 3/8 = 5/8$
  - $e(X) = (||\pi'_X|| - |\pi'_X|)/r$
    - $||\pi'_X||$ = sum of sizes of equivalence classes in π'
    - $e(B) = ((5+2) - 2)/r = 5/8$
  - $X \rightarrow A \Leftrightarrow e(X) = e(X \cup A)$

	A	B	C
1	1	A	a
2	1	A	b
3	2	A	c
4	2	A	c
5	2	A	d
6	3	B	e
7	3	B	e
8	3	D	f

Computing partitions

- Compute partition π_A for each $A \in R$
  - Directly from database
  - Only store tuples ID (Integers)
- Product $\pi 1 \cdot \pi 2$: Least refined partition that refines both $\pi 1$ and $\pi 2$
  - $\pi_X \cdot \pi_Y = \pi_{X \cup Y}$
  - Partitions π_X computed as product of two partitions of size $|X|-1$.
  - Algorithm on next slide
- Dependency checking: $X \backslash A \rightarrow A$
  - Calculate $e(X) = (||\pi'_X|| - |\pi'_X|)/r$ and $e(X \backslash A) = \ldots$
  - Check $e(X) = e(X \backslash A)$
- Also key pruning: X is key if $e(X) = 0$.

Partition Product

Input: Stripped partitions $\widehat{\pi'} = \{c'_1, \ldots, c'_{|\widehat{\pi'}|}\}$ and $\widehat{\pi''} = \{c''_1, \ldots, c''_{|\widehat{\pi''}|}\}$.

Output: Stripped partition $\widehat{\pi} = \widehat{\pi' \cdot \pi''}$.

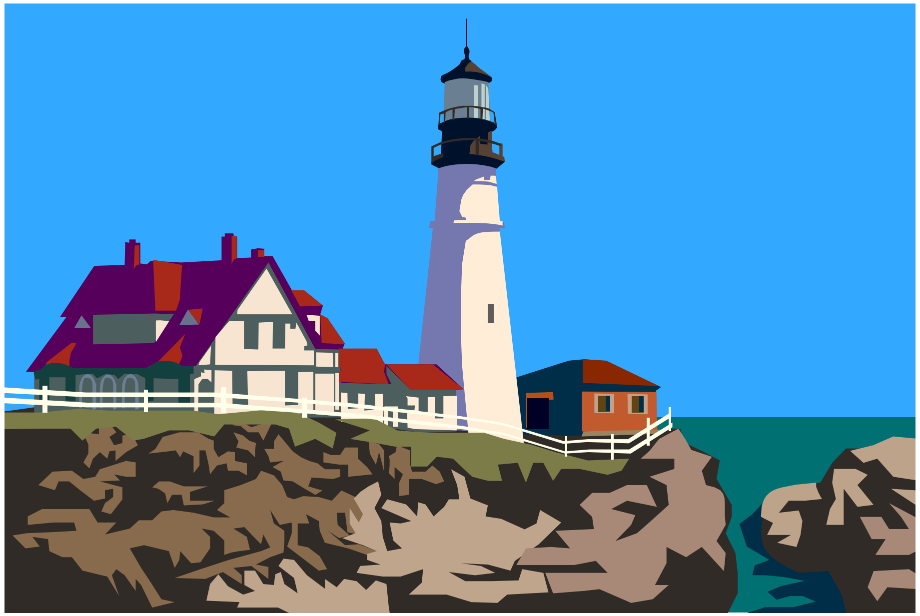
$\widehat{\pi} := \emptyset$
for $i := 1$ **to** $|\widehat{\pi'}|$ **do**
  for each $t \in c'_i$ **do** $T[t] := i$
  $S[i] := \emptyset$
for $i := 1$ **to** $|\widehat{\pi''}|$ **do**
  for each $t \in c''_i$ **do**
    if $T[t] \neq \text{NULL}$ **then** $S[T[t]] := S[T[t]] \cup \{t\}$
  for each $t \in c''_i$ **do**
    if $|S[T[t]]| \geq 2$ **then** $\widehat{\pi} := \widehat{\pi} \cup \{S[T[t]]\}$
    $S[T[t]] := \emptyset$
for $i := 1$ **to** $|\widehat{\pi'}|$ **do**
  for each $t \in c'_i$ **do** $T[t] := \text{NULL}$
return $\widehat{\pi}$

	A	B	C
1	0	4	7
2	1	5	7
3	1	5	8
4	2	6	9
5	2	6	9
6	3	4	7

$\pi'_{A \cup B} = \{23, 45\}$
$\pi'_{B \cup C} = \{16, 45\}$

Overview

- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



Making TANE approximate

- Definition based on minimum number of tuples to be removed from R for $X \rightarrow A$ to hold in R.
- Discovery problem:
  - Given relation R and threshold ε, find all minimal non-trivial FDs $X \rightarrow A$ such that $e(X \rightarrow A) \leq \varepsilon$.

1. Define error: Fraction of tuples causing FD violation
   - Error $e(X \rightarrow A) = \min\{|S| \mid S \subseteq R,\ R \setminus S \vDash X \rightarrow A\} / |R|$
2. Specify error threshold ε
3. Modify dependency checking algorithm
   - Efficient algorithm to compute error
   - Bounds to avoid error calculation

Approximate Dependency Checking

Exact version

1 **for each** $X \in L_\ell$ **do**

2 $\quad C^+(X) := \bigcap_{A \in X} C^+(X \setminus \{A\})$

3 **for each** $X \in L_\ell$ **do**

4 $\quad$ **for each** $A \in X \cap C^+(X)$ **do**

5 $\quad\quad$ **if** $X \setminus \{A\} \rightarrow A$ is valid **then**

6 $\quad\quad\quad$ output $X \setminus \{A\} \rightarrow A$

7 $\quad\quad\quad$ remove A from $C^+(X)$

8 $\quad\quad\quad$ remove all B in $R \setminus X$ from $C^+(X)$

Modifications

5′ $\quad\quad$ **if** $e(X \setminus \{A\} \rightarrow A) \leq \varepsilon$ **then**

8′ $\quad\quad\quad$ **if** $X \setminus \{A\} \rightarrow A$ holds exactly **then**

9′ $\quad\quad\quad\quad$ remove all B in $R \setminus X$ from $C^+(X)$

Computing error

- Error $e(X \rightarrow A) = \min\{|S| \mid S \subseteq R,\ R \setminus S \models X \rightarrow A\} \ / \ |R|$
- Any equivalence class $c \in \pi_X$ is the union of one or more equivalence classes c1‘,c2‘, …$\in \pi_{X \cup A}$
- For each $c \in \pi_X$ the tuples in all but one of the c_i‘s must be removed for X→A to hold.
- Minimum number to remove: Size of c minus size of largest c_i’.
- $e(X \rightarrow A) = 1 - \frac{\sum_{c \in \pi_X} \max\{|c'| \mid c' \in \pi_{X \cup A} \land c' \subseteq c\}}{|R|}$
- Example: B→A
  - π_A = {12, 345, 678}
  - π_B = {1, 234, 56, 78}
  - π_{AB} = {1, 2, 34, 5, 6, 78}
  - $|\pi_B| \neq |\pi_{BA}|$
  - e(B→A) = 8/8 – (1+2+1+2)/8 = 2/8
- Also possible on stripped partitions – not here.

	A	B
1	1	a
2	1	A
3	2	A
4	2	A
5	2	b
6	3	b
7	3	C
8	3	C

Bounding on error

- Computing error is in O(|R|)

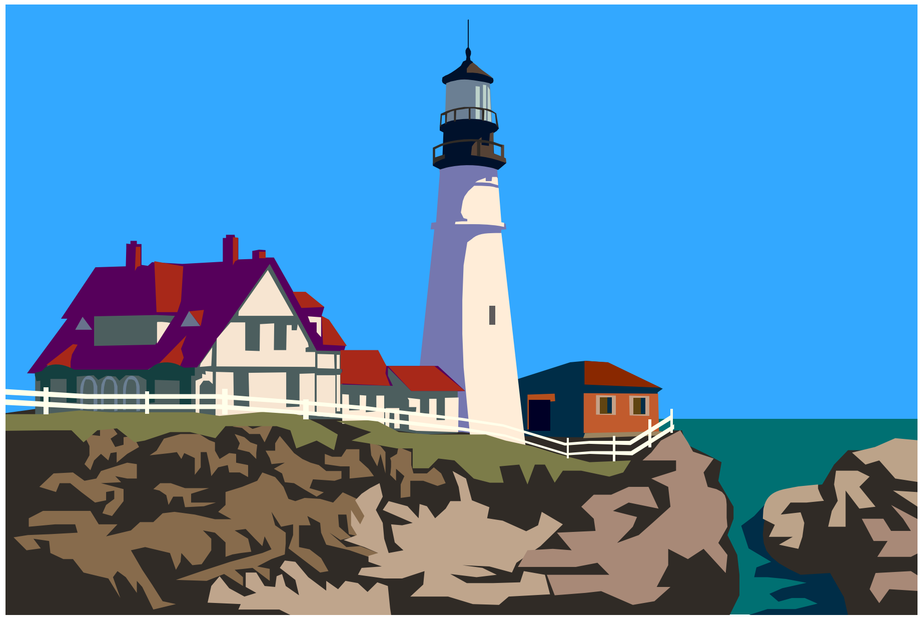
- $e(X) - e(X \cup A) \le e(X \rightarrow A) \le e(X)$
- I.e., do not calculate FD error if
  - $e(X) - e(X \cup A) > \varepsilon$
  - $e(X) < \varepsilon$

- $e(B) = 4/8$
- $e(B \cup A) = 2/8$
- $e(B \rightarrow A) = 8/8 - (1+2+1+2)/8 = 2/8$

	A	B
1	1	a
2	1	A
3	2	A
4	2	A
5	2	b
6	3	b
7	3	C
8	3	C

Overview

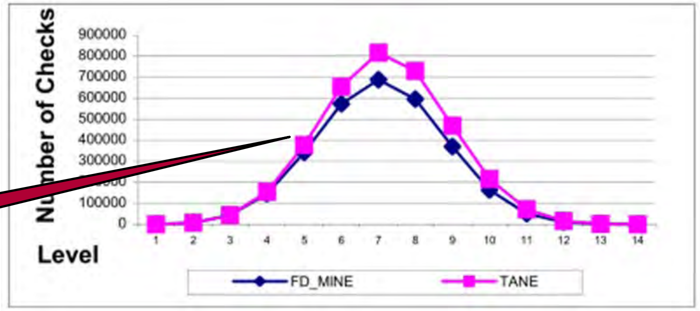
- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



FD_Mine: Refinement of TANE

- If X → Y and Y → X hold, then X and Y are *equivalent candidates*, denoted as X↔Y.
- Make use of additional FD properties
  - X↔Y and XW → Z ⇒ YW → Z
  - X↔Y and WZ → X ⇒ WZ → Y
- Example
  - A→D and D→A ⇒ A↔D
  - Examination: AB → C and BC → A
  - Inferred:
    - BD → C (property 1)
    - BC → D (property 2)
  - D can be removed from table

	A	B	C	D	E
1	0	0	0	1	0
2	0	1	0	1	0
3	0	2	0	1	2
4	0	3	1	1	0
5	4	1	1	2	4
6	4	3	1	2	2
7	0	0	1	1	0

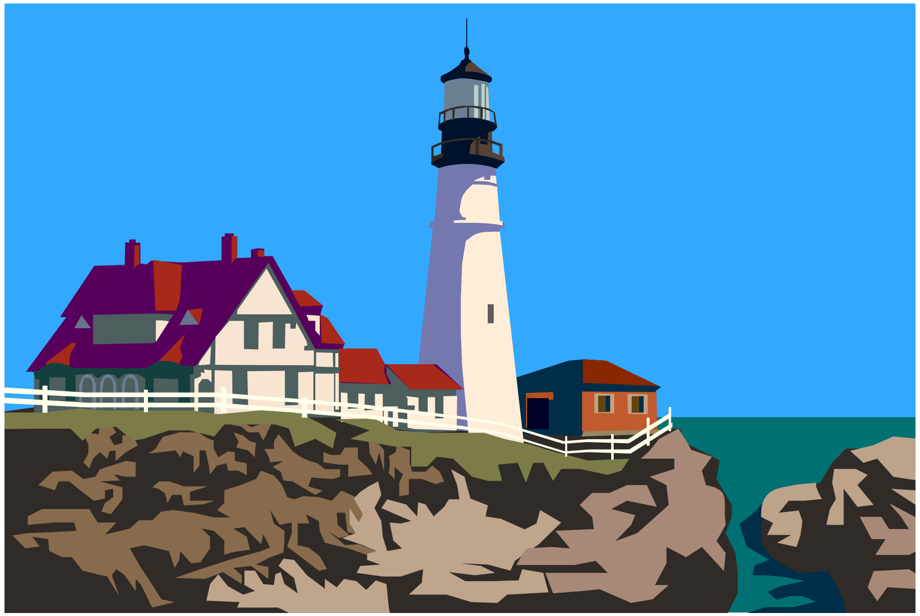


Pairs of UCCs

- Hong Yao, Howard J. Hamilton, Cory J. Butz: FD_Mine: *Discovering Functional Dependencies in a Database Using Equivalences.* ICDM 2002: 729-732

Overview

- Functional Dependencies
- TANE
  - Candidate sets
  - Pruning Algorithm
  - Dependency checking
  - Approximate FDs
- FD_Mine
- Conditional FDs



Conditional Functional Dependencies

- Idea similar to CINDs
- Embedded FD plus pattern tableau
- Definition CFD
  - Pair $(X \rightarrow A, T_p)$
    - $X \rightarrow A$ is embedded FD
    - T_p is pattern tableau, made up of pattern tuples t_p
  - Pattern tuple with attributes $B \in X \cup A$ where $t_p[B]$
    - Constant in dom(B)
    - Unnamed variable "_" for values in dom(B)
  - Special case: $T_p[B]$ = "_" for all B is equivalent to normal FD

Semantics of CFDs

- $a \approx b$ (a matches b) if
  - either a or b is "_"
  - both a and b are constants and $a = b$

- DB satisfies $(R: X \rightarrow Y, T_p)$ iff
  - For any tuple t_p in the pattern tableau T_p and for any tuples t_1, t_2 in DB:
    - If $t_1[X] = t_2[X] \approx tp[X]$, then $t_1[Y] = t_2[Y] \approx t_p[Y]$
  - $t_p[X]$: identifying the set of tuples on which the constraint t_p applies: $\{ t \mid t[X] \approx t_p[X] \}$
  - $t_1[Y] = t_2[Y] \approx t_p[Y]$: enforcing the embedded FD, and the pattern of t_p

Slide from Wenfei Fan

Example: Violation of CFDs

id	country	area-code	phone	street	city	zip
t1	44	131	1234567	Mayfield	NYC	EH4 8LE
t2	44	131	3456789	Crichton	NYC	EH8 8LE
t3	01	908	3456789	Mountain Ave	NYC	07974
t4	01	908	9876543	Mainstreet	NYC	07974

country	zip	street
44	_	_

- cust([country, zip] → [street], Tp)
- Tuples t1 and t2 violate the CFD
  - t1[country, zip] = t2[country, zip] ≈ t_p[country, zip]
  - But t1[street] ≠ t2[street]
- The CFD applies to t1 and t2 since they match t_p[country, zip]
- Tuples t3 and t4 do not violate the CFD
  - CFD does not apply to t3 and t4

Slide from Wenfei Fan

Example: Violation of CFD by single tuple

id	country	area-code	phone	street	city	zip
t1	44	131	1234567	Mayfield	NYC	EH4 8LE
t2	44	131	3456789	Crichton	NYC	EH8 8LE
t3	01	908	3456789	Mountain Ave	NYC	07974
t4	01	908	9876543	Mainstreet	NYC	07974

- cust([country, zip] → [street], Tp)
- Tuple t1 does not satisfy the CFD.

	country	zip	street
tp1	44	131	Edi
tp2	01	908	MH
tp3	_	_	_

- t1[country, area-code] = t1[country, area-code] ≈ tp1[country, area-code]
- t1[city] = t1[city]; however, t1[city] does not match tp1[city]
- In contrast to traditional FDs, a single tuple may violate a CFD.

Slide from Wenfei Fan

Further literature

- DepMiner
  - *Efficient Discovery of Functional Dependencies and Armstrong Relations*. Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. EDBT 2000
- CORDS
  - Ihab F. Ilyas, Volker Markl, Peter J. Haas, Paul Brown, Ashraf Aboulnaga: CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. SIGMOD Conference 2004: 647-658
- FastFDs
  - Catharine M. Wyss, Chris Giannella, Edward L. Robertson: *FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances*. DaWaK 2001: 101-110
- CFDs
  - Loreto Bravo, Wenfei Fan, Shuai Ma: Extending Dependencies with Conditions. VLDB 2007: 243-254