**CSCI 4333.2**

11/11/2024

**Theory of Functional Dependency**

by K. Yue

**1. Armstrong's axioms**

* Armstrong's axioms is an inference system.
* Reasoning from the *definition*is difficult.

A relation *schema* R is said to *satisfy* the *functional dependency* X -> Y if for *any* relation *instance* r that uses R, if there exists two tuples s and t ∈ r such that s[X] = t[X], then s[Y] = t[Y].

1. (∃s, t ∈ r) (s[X] = t[X]) => s[Y] = t[Y]
2. i.e.  same value in X implies same value in Y.
* Thus, people invented axioms as an equivalent way for*inference*.
* A set of *axioms* for inference with FD: <http://en.wikipedia.org/wiki/Armstrong%27s_axioms>.
* Axioms: 'self-evidence' or 'assumed' so that they can be used as the basis of inference.
* Three basic axioms:
	1. *Reflexivity:* If X and Y are sets of attributes and Y is a subset of X, then X -> Y.
	2. *Augmentation*: If X -> Y then X Z -> Y Z (LHS and RHS are both augmented by Z)
	3. *Transitivity***:**If X -> Y and Y -> Z then X -> Z
* Three additional rules that can be derived from the basic axioms.
	1. *Pseudo-transitivity* Rule: If X-> Y, YZ -> A then XZ -> A
	2. *Decomposition* Rule: If X -> Y Z, then X -> Y and X -> Z. (Note that the decomposition applies to the RHS of the FD)
	3. *Union* Rule:  If X -> Y and X -> Z then X -> Y Z. (Note that the decomposition applies to the RHS of the FDs)
* Armstrong's axioms are *sound* and *complete*.
	1. Soundness: Inference using the axioms will create only correct FD.
	2. Completeness: The axioms can be used to derive *all* correct FD.
* Computing students need to know how to infer using a formal mathematical method.

***Example***

Let X be CITY STREET, Y be STREET, then Y is a subset of X, and X -> Y or CITY STREET -> STREET (Reflexivity axiom).

* If two tuples have the same values of CITY and STREET, then they surely have the same value of STREET.
* This is so trivial that we call a functional dependency likes CITY, STREET -> STREET a *trivial functional dependency*. They do not actually specify any problem requirement but are mathematical true.

***Example:***

For R(A,B), we have the following trivial FD for the attributes A and B. No matter what A and B are supposed to mean, they are always mathematically true. (Φ is the empty set.)

AB -> AB, AB->A, AB->B, AB-> Φ
A -> A, A-> Φ
B -> B, B-> Φ

Remember AB-> AB means {A, B} -> {A, B}.

* Since trivial functional dependencies do not actually represent any problem requirements, we are only interested in *non-trivial* functional dependency. Non-trivial FD are FD in which its RHS is not a subset of its LHS. (not universal, present requirement)

If EmpId  ->  DeptId, and DeptId  ->  ManagerId
then EmpId  ->  ManagerId.

Interpretation: If

1. every Employee works for only one department, and
2. every department has only one manager,

then every Employee has only one manager.

A non-trivial FD X->Y:

1. Y is not a subset of X.
2. It represents problem requirements.
3. It is not universally true. It may be false under a different set of problem requirements.

A mathematical proof using Armstrong's axiom is to continuously create/prove new FDs until the result is included. Reasons are usually given.

***Example***

Prove that the decomposition rule is true: X->YZ => X->Y and X->Z

Proof:

[1] X->YZ (given)
[2] YZ -> Y (reflexivity axiom)
[3] X -> Y (transitivity axiom on [1] and [2]).
[4] YZ -> Z (reflexivity axiom)
[5] X -> Z (transitivity axiom on [1] and [4]).

**2. Keys and Superkeys Revisited**

* We can use the concept of FD to define keys and superkeys.
* For a relation scheme R, K is a candidate key (CK) if
	1. Uniqueness:  K -> R.
	2. Minimality:  there is no *proper* subset of K that determines R. (There is no *extraneous* attribute.)
* Note that
	1. |A| = cardinality of A = the number of elements in the set A.
	2. A ⊆ B means A is a subset of B and it is possible that A = B.
	3. A ⊂B means A is a proper subset of B in which A <> B.
	4. If A ⊆ B, |A| <= |B|.
	5. If A ⊂ B, |A| < |B|.
* K is a superkey if K -> R.
* Superkeys (SK) do not need to satisfy the minimality requirement.
* Some properties:
	1. If K is a CK, any superset of K is a SK.
	2. If K is a CK, any proper subset of K is not a CK (not unique)
	3. If K is a CK, any proper superset of K is not a CK (not minimal).
* Note that the primary key of a table is just a selected candidate key used to structure the p*hysical storage*. PK has the same logical properties like other candidate keys (*alternate or secondary keys*) in the context of the normalization theory.
* A CK with only one attribute is known as a *simple key*.
* A CK with more than one attributes is known as a *composite key*.
* A *compound* key is a composite key in which every component attribute is a foreign key.

***Example***

In Employee(EmpId, DeptId, ManagerId) with

EmpId -> DeptId, and
DeptId -> ManagerId.

By the transitivity axiom, EmpId -> ManagerId
By the union rule, EmpId -> EmpId, DeptId, ManagerId
By the augmentation axiom, EmpId, ManagerId -> DeptId, ManagerId

Hence, EmpId is a CK of Employee(EmpId, DeptId, ManagerId).

On the other hand,

1. DeptId is not a candidate key since we do not have DeptId -> EmpId.
2. {Empd, DeptId} is not a candidate since it is not minimal. It is a superkey only.

Furthermore, there are four superkeys:

1. EmpId
2. EmpId, DeptId
3. EmpId, ManagerId
4. EmpId, DeptId, ManagerId

**3. Finding Candidate Keys**

**3.1 Closure of Attributes**

* Given a set of FD F, the *closure* of a*set of attributes* X, denoted as X+, is the set of all attributes functionally determined by X using Armstrong's axioms on F.

***Example***

Consider R(A,B,C,D) with

F = {B->A, A->C, AB->D, D->AC}

A+: A
 : A C

A+ = AC

F = {B->A, A->C, AB->D, D->AC}

B+: B
 : B A
 : B A C
 : B A C D = R

B is a CK of R

B+ = ABCD = R
C+ = C

F = {B->A, A->C, AB->D, D->AC}

D+: D
 : D AC
D+ = ACD

Thus, B is a candidate key (CK).

No proper superset of B is a candidate key, e.g. BA , BC, BD, BAC, BAD, BCD, ABCD, is a CK (since it will not be minimal).

Remaining non-empty subsets of ABCD to check for candidate keys:

AC+ = AC
AD+ = ACD
CD+ = ACD
ACD+ = ACD

Thus, B is the only CK.

* The closure of attributes can be used for other purposes, such as checking validity of FD, computing closures of a set of functional dependencies, checking equivalence of two set of FDs, etc.

**3.2 Algorithm for finding X+ for a set of FDs F.**

[1] X+ <- X // Start with X in X+ because X -> X.
[2] while (
         [A] there exists a FD P -> Q such that
         [B] P is a subset of X+, and
         [C] there are attributes K in Q not in X+) {
   [3] X+ <- X+ U Q      // Add attributes in Q to X+ by using the union operator.
}

**3.3 Finding Candidate keys**

* It is necessary to find *all* candidate keys to conduct normalization analysis.
* In general, if R has n attributes, there are 2n - 1 subsets of R which are potential candidate keys. NP-Complete problem.

***Example:***

For R(A,B,C), need to check A, B, C, AB, AC, BC and ABC for candidate keys.

Thus, the problem of finding all candidate keys in R is O(en), where n is the number of attributes in the relation R.

**3.4 To find all candidate keys of R with a set of FD, F:**

1. Find the *canonical cover*, FC, first. This simplifies F. (See later)
2. Use *heuristics* to cut down the number of sets of attributes to check for candidate keys.
3. Classify attributes into three groups:
	1. L/NR (left only or not right): If an attribute X does not appear in the right hand side (RHS) of any f in F, *every* candidate key must include X.
	2. R (right only): If X appears only in the RHS of a fd in F but does not appear in the LHS of any f in F, then x is *not* a component of *any* candidate key.
	3. M (mixed; left and right): If X appears in LHS in some FD and in RHS in some other FD in F, then X *may* potentially be in some CK.
4. If X is found to be a CK, then any proper superset of X is not a CK, and needs not be checked.

***Example:***

Consider R(A,B,C,D) with

F = {B->A, A->C, AB->D, D->AC}

We have:

L/NR: B (in every CK)
M: A, D (may be in some CK)
R: C (not in any CK)

Checking: B and then BA, BD, BAD (if needed).

B+: BACD

Thus, there is only one CK: [1] B.

[4] (20%) Consider the following relation R(A,B,C,D,E) {A->B, AB->D, AD->E, C->D} (a) Show all candidate keys. (b) What is the highest normal form (up to BCNF)? Why? (c) If it is not in BCNF, can you losslessly decompose R into component relations in BCNF while preserving functional dependencies?

R(A,B,C,D,E) {A->B, AB->D, AD->E, C->D}

L/NR: A, C
M: B, D
R: E

Check AC

AC+: AC
 : AC B
 : AC B D
 : AC B D E = R

CK: [1] AC
prime (key) attributes: A, C (in some CK)
non-prime attributes: B, D and (in non CK)

**4. FD Closure and Covers (If time permits)**

**4.1 Closure of a set of functional dependencies (FD)**

* The *closure* of a set of FD, F, is denoted by F+, and is the set of all FDs that are*logically implied* by F.

Consider F = {A->B, B->C}

F+ = {
A->{}, A->A, A->B, A->C, A-> AB, A-> AC, A-> BC, A->ABC,
B->{}, B->B, B->C, B->BC,
C->{}, C->C,
AB->{}, AB->A, AB->B, AB->C, AB->AB, AB->AC, AB->BC, AB->ABC,
AC->{}, AC->A, AC->B, AC->C, AC->AB, AC->AB, AC->BC, AC->ABC,
BC->{}, BC->B, BC->C, BC->BC,
ABC->{}, ABC->A, ABC->B, ABC->C, ABC-> AB, ABC-> AC, ABC-> BC, ABC->ABC }

Note that

* Many FDs in F+ are trivial. Examples: A->{}, ABC->AC, etc.
* FD+ itself is not very interesting.

**4.2 Equivalence and cover**

* Two sets of FD, F and G are *equivalent*, if F+ = G+. They are *covers* of each other.
* Thus, covers can be used to support the concepts of equivalence.
* If F and G are covers of each other, they represent the same set of application requirements and assumptions.

**4.3 Canonical and Minimal Covers**

* **Definition.** In a set of FDs F, the attribute A in the FD P-> Q is *extraneous* if F - {P-> Q} U {P-A -> Q} is equivalent to F.
* Thus, the attribute A is not actually needed in P to determine Q. It is extraneous.

***Example***

Consider the F = {A->B, AB->C}.

B is extraneous since for G = {A->B, A->C}, and F+ = G+.

* **Definition**. A FD f in F is *redundant* if (F - f)+ = F+.

***Example***

In F = {A->B, AB->C, B->C},

AB->C is redundant since for

G = {A->B, B->C}, AB+ = ABC.

Alternatively, we may state that

G |- AB-> C.

***Example***

For F = {A->BC, B->C}

Using decomposition rule,

F' = {A->B, A->C, B->C} is a cover of F.

In F', A->C is redundant since {A->B, B->C} |- A->C

Thus F" = {A->B, B->C} is a cover of F' and F.

* **Definition.** A *canonical cover*, G, of F satisfies the following conditions:
	1. G is a cover of F; G+ = F+.
	2. There is no redundant FD in G.
	3. There is no extraneous attribute in G.
	4. The left hand side (LHS)of every FD in G is unique.
* **Definition.** A *minimal cover*, G, of F satisfies the following conditions:
	1. G is a cover of F; G+ = F+.
	2. There is no redundant FD in G.
	3. There is no extraneous attribute in G.
	4. The right hand side (RHS) of every FD in G contains only a single attribute

In F = {A->B, AB->C, B->C, A->D},

G1 = {A->B, B->C, A->D} is a minimal cover.

G2 = {A->BD, B->C} is a canonical cover.

* The minimal covers and canonical covers are *simplified* equivalent versions of a set of FDs, representing the same set of data constraints.
* They are useful in understanding FD and for proper decompositions to remove unnecessary redundancy.

***Example:***

Consider F: {A->C, BCD->A, C->E, CD-> A, AB->C}

[1] Does F imply BD-> A (i.e. F |- BD -> A)?

No, Since in F, BD+ = BD

Thus, C is not extraneous in BCD -> A.

[2] F |- AE -> B ?

No, since AE+ = AE C

[3] Give a canonical cover for F.

{ A->C, CD->A, C->E }

[4] Show all candidate keys.

L/NR: B, D
M: A, C
R: E

CK: [1] ABD, [2] CBD

***Example (Tedious):***

Find a canonical cover for F = {BC->AE, AD->BCE, A->E, AE->D, BCD->F, AB->C}

***Solution:***

Basically, we iteratively remove all extraneous attributes and redundant function dependencies.

We use decomposition rule to ensure the RHS to contain only a single attribute so we can work on them one by one. F becomes:

(1) BC -> A
(2) BC -> E
(3) AD -> B
(4) AD -> C
(5) AD -> E
(6) A -> E
(7) AE -> D
(8) BCD -> F
(9) AB -> C

To investigate whether B or C is extraneous in BC -> A, we note that in F:

B+ = B
C+ = C

This means B alone and C alone cannot determine A, and neither of them is extraneous.

On the other hand, in F:

A+ = ABCDEF

That means A alone can determine all other attributes. Any other attributes in the LHS with A in a FD are thus extraneous, we thus have the following by removing D in [2], [3] and [4], and B in [9].

(1) BC -> A
(2) BC -> E
(3) A -> B
(4) A -> C
(5) A -> E
(6) A -> E
(7) A -> D
(8) BCD -> F
(9) A -> C

Removing identical FD, we have F:

(1) BC -> A
(2) BC -> E
(3) A -> B
(4) A -> C
(5) A -> E
(6) A -> D
(7) BCD -> F

For (7), since B+ = B, C+ = C and D+ = D. However, BC+ = ABCDEF, and thus D is extraneous. Thus, we now have:

(1) BC -> A
(2) BC -> E
(3) A -> B
(4) A -> C
(5) A -> E
(6) A -> D
(7) BC -> F

To check for redundant FD, we consider whether we can deduce the FD when it is removed.

For (1) BC -> A, removing it result in F':

(1) BC -> E
(2) A -> B
(3) A -> C
(4) A -> E
(5) A -> D
(6) BC -> F

In F': we have

BC+ = BCE, which does not include A. Thus, F' does not imply BC -> A and it is not redundant.

For (2) BC -> E, removing it and we have F':

(1) BC -> A
(2) A -> B
(3) A -> C
(4) A -> E
(5) A -> D
(6) BC -> F

In F', we have BC+ = ABCDEF. Thus, F' |= BC -> E and BC -> E is redundant. Remove it and we have:

(1) BC -> A
(2) A -> B
(3) A -> C
(4) A -> E
(5) A -> D
(6) BC -> F

Using this method, we can find that there are no more redundant FD.

Finally, we use the union rule to merge FD with the same LHS and get the canonical cover:

{BC -> AF, A-> BCDE}

Note that the canonical cover is not unique. Another canonical cover is:

{BC -> A, A-> BCDEF}

***Exercise:***

Consider F: {AB->CE, BC->D, D->BC, C->E, A->C, A->E}

Find:

* all candidate keys.
* a canonical cover of F.

***Exercise:***

Can there be more than one canonical covers for a set of FDs?