**FUNCTIONAL DEPENDECIES**

The fundamental tool for normalization theory
**********
* - May seem dry and irrelevant, but bear with me. Extremely useful
* - Things to learn
* - trivial FD, logical implication, closure, FD and key, projected FD
**********

Formal Properties:

TRIVIAL functional dependency: X -> Y where Y SUBSET X: always true

NON-TRIVIAL FD: X -> Y is not a subset of X

COMPLETELY NON-TRIVIAL FD: X -> Y with no overlap between X and Y

We will focus on completely non-trivial functional dependency>

LOGICAL IMPLICATION:

Given R(A, B, C, G, H, I) and a set of functional dependencies in R:

A -> B
A -> C
CG -> H
CG -> I
B -> H

Can we infer that

Q: A -> H also hold ?

Inference rules:

1. **Reflexivity:** If Y is a nonempty subset of X then X -> Y.
2. **Augmentation:** if X -> Y then X ? Z -> Y ? Z.
3. **Transitivity:** X -> Y and Y->Z then X->Z.

This set is sound and complete.

CLOSURE OF a set of FDs F: F+
`

F+ : the set of all FD's that are logically implied by F.

CLOSURE (w.r.t. to F) OF ATTRIBUTE SET X:
Id denoted X+

X+: the set of all attrs functionally determined by X (A in X+ iff X->A in F+)

Q: What attributes do we know given (sid, dept, cnum)+?

An Efficient Algorithm for closure computation:

**start with X+ = X**
**repeat until no change in X+**
**if there is Y -> Z and Y is a subset of X+, add Z to X+**

<example>
A -> B
A -> C
CG -> H
CG -> I
B -> H

Q: {A}+?

Q: {A,G}+?

**Normal Form Design**

FUNCTIONAL DEPENDENCY AND KEYs
- a key determines a tuple
- functional dependency determines other attributes

X is a KEY of R iff

1) X -> all attributes of R (ie, X+ = R)
2) No subset of X satisfies 1) (i.e., X is minimal)

Q: In previous example, is (A, B) a key of R?

PROJECTING FDs: R(A, B, C, D): A -> B, B A, A-> C

Q: What FDs hold for R'(B, C, D) which is a projection of R?

Information on FDs might be lost unless:

1. we first compute F+ : e.g. A ->B, B->A, B->C
2. we select the right decomposition: A ->B, B->C

Preservation of FD information is an important objective … another

**********
* LOSSLESS-JOIN DECOMPOSITION
**********
<e.g. splitting StudentClass table to make it better>

DECOMPOSITION
- Split table R(A1, ..., An) into R1(A1, ..., Ai) and R2(Aj, ..., An)
- {A1, ... An} = {A1, .., Ai} UNION {Aj, ..., An}

<Conceptual diagram for R(X, Y, Z) -> R1(X, Y) and R2(Y, Z)

Q: When we decompose, what should we watch out for?

LOSSLESS-JOIN DECOMPOSITION
- R = R1 NJ R2
- Intuitively, we should not lose any information by decomposing R
- Can reconstruct the original table from the decomposed table

Q: When is decomposition lossless?

<example>

| cnum | sid | name |
|---|---|---|
| 143 | 1 | James |
| 143 | 2 | Elaine |
| 325 | 3 | Susan |

<eg1, Decompose into S1(cnum, sid), S2(cnum, name). Lossless?>
<Explain using the example>

- Main problem:
  - A tuple in S1 may join with two tuples in S2
  - A tuple in S2 may join with two tuples in S1
    - Tuples multiply because of these

<eg2, Decompoes into S1(cnum, sid), S2(sid, name). Lossless?>
<Explain using the example>

- Main reason for lossless:
  - Every S1 tuples joins with exactly one S2 tuple
  - Common attribute, sid, uniquely determines a tuple in S2
    - due to FD, sid -> name

** DECOMPOSITION R(X, Y, Z) => R1(X, Y), R2(X, Z) IS LOSSLESS IF X->Y OR X->Z **
- i.e., The shared attributes are the key of one of the decomposed tables
- We can use FDs to check whether a decomposition is lossless

<e.g., StudentClass(sid, name, addr, dept, cnum, title, unit)>
sid->(name,addr) (dept,cnum)->(title,unit)

Q: R1(sid, name, addr), R2(sid, dept, cnum, title, unit). Lossless?

**********
* FD, KEY & REDUNDANCY
**********

StudentClass(sid, name, addr, dept, cnum, title, unit)
Q: sid->(name,addr). Does it cause redundancy?

After decomposition, Student(sid, name, addr)
Q: sid->(name,addr). Does it still cause redundancy?

Q: Why does the same FD cause redundancy in one case, but not in the other?

A: Main difference
- sid is a key for the second example, but not for the first.
- Because sid is a key, (sid, name, addr) is stored only once in the second
  -> no redundancy
- Because sid is not a key, (sid, name, addr) may be stored multiple times in the first.

IN GENERAL, FD X->Y IS BAD IF X DOES NOT CONTAIN A KEY.

**********
* BOYCE-CODD NORMAL FORM (BCNF)
**********
DEFINITION:
- R is in BCNF with regard to F, iff for every non-trivial X -> A,

X contains a key

- a "good" table design (no redudancy due to FD)

Q: S(dept, cnum, title, unit). BCNF?
dept,cnum->title,unit

Q: S(instructor, office, fax). BCNF?
instructor->office
office->fax

BCNF NORMALIZATION: decompose tables until all tables are in BCNF
- For each FD that violates the condition, use it to decompose the table and remove the redundancy from it.
- In addition, we want to make sure that the decomposition is lossless.
-> The algorithm is designed to guarantee this

DECOMPOSITION ALGORITHM for BCNF

For any R in the schema:

If non-trivial X -> Y holds on R, and if X does not have a key
1) Compute X+ (X+: closure of X)
2) Decompose R into R1(X+) and R2(X, Z) // X is common attributes
where Z is all attributes in R except X+

Repeat until no more decomposition:

<eg, ClassInstructor(dept, cnum, title, unit, instructor, office, fax)>
F: instructor -> office
office -> fax
(dept, cnum) -> title, unit
(dept, cnum) -> instructor

Some nice properties of the BCNF algorithm:

1. minimizes the relation count by avoiding decomposing relations with equivalent keys:
R(A, B, C, D) where A->B, A->C.

2. Preserves FD information: <Example …> In most cases …. (more about that later)

3. The algorithm guarantees lossless join decomposition. After decomposing on X->Y, X becomes the key of one of the decomposed table

<example>
R(A, B, C, G, H, I)
A -> B
A -> C
G -> I
B -> H

Convert to BCNF

Q: Does the algorithm lead to a unique set of relations?

<eg., R(A, B, C) A->C, B->C>
If we start with A->C: R1(A, C), R2(A, B)
If we start with B->C: R1(B, C), R2(A, B)

Q: R1(A, B), R2(B, C, D)
A->B
B->A
A->C
Are R1 and R2 in BCNF?

COMMENTS: We have to check all implied FD's for BCNF, not just the given ones.

**********
* DEPENDENCY-PRESERVING DECOMPOSITION
**********
- FD is a kind of constraint.
- We sometimes may want to make sure that an update does not violate FDs.

<eg, R(office, fax), office -> fax. How do we enforce the FD?>

<eg, R1(A, B), R2(A, C), B->C. How to enforce it?>

- FD violation checking can be very expensive.
- Can we make sure we can enforce FD's without joins?

<eg, R1(A, B), R2(B, C), A->B, B->C, A->C. How to enforce them?>

- No need to check A->C directly (A->B, B->C implies A->C)
- local "FD" checking possible

DEPENDENCY-PRESERVING DECOMPOSITION:
After decomposition, we can enforce FD's LOCALLY on each tables without joins.

Q: Does BCNF normalization lead to dependency-preserving decomposition?

<eg, ClassInstructor(dept, cnum, title, unit, instructor, office, fax)>
F: instructor -> office
office -> fax
(dept, cnum) -> title, unit
(dept, cnum) -> instructor

=== BCNF normalization ===>

R1(dept, cnum, title, unit, instructor), R2(instructor, office),
R3(office, fax)

Q: Is it dependency preserving?

Q: Is it generally true?

<eg, R(street, city, zip), (street,city)->zip, zip->city>

==> R1(street, zip), R2(zip, city).

Q: Is it dependency preserving?

Q: Is there an alternative definition of normal form that guarantees dependency preservation?

**********
* THIRD-NORMAL FORM (3NF)
**********
- R is in 3NF with regard to F, iff for every non-trivial X -> A either
(1) X contains a key OR

(2) a is a member of some key

- An alternative normal form that allows dependency preservation
  - Do not ask me why. I do not have the intuition for the second condition.
  - minimal relaxation for dependency preservation.

<eg., R(street, city, zip)>
(street,city) -> zip,  zip -> city

Q: BCNF?

Q: 3NF?

THEOREM: We can always decompose a relation into 3NF relations and check all FD's locally.

- It is not very intuitive, but the second condition
  "A  is a member of some key" is the "MINIMAL RELAXATION" of BCNF to allow dependency preservation.

NOTE: Similarly to BCND decomposition algorithm, there exists "3NF Synthesis algorithm" that decomposes a table into a set of 3NF tables that preserve dependency.

COMPARISON OF 3NF and BCNF

Q: If R is in BCNF, is it in 3NF?

Q: If R is in 3NF, is it in BCNF?

BCNF
- removes all redundancy
- does not guarantee dependency-preserving decomposition

3NF
- dependency-preserving decomposition always possible
- does not remove all redundancy

In practice,  the cases where the result produced by the BCNF design algorithm do preserve dependencies are so special that they might require ad-hoc treatment---e.g., 3NF with redundancy, and/or BCNF with triggers ….

Timely Progression:  1NF , 2NF, 3NF, BCNF (using FDs) …, 4NF (using MVDs)