**Normal Forms**

o A set of rules to avoid redundancy and inconsistency.

o Require the concepts of

o functional dependency (most important: up to BCNF)

o multivalued dependency (4NF)

o join dependency (5NF)

o Seven Common Normal Forms: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF, DKNF.

+------------------------------------------------------+

| All relations (normalized or un-normalized) |

| +-----------------------------------------------+ |

| | 1NF relations | |

| | +----------------------------------------+ | |

| | | 2NF relations | | |

| | | +----------------------------------+ | | |

| | | | 3NF relations | | | |

| | | | +----------------------------+ | | | |

| | | | | BCNF relations | | | | |

| | | | | +----------------------+ | | | | |

| | | | | | 4NF relations | | | | | |

| | | | | | +-----------------+ | | | | | |

| | | | | | | 5NF relations | | | | | | |

| | | | | | | +-----------+ | | | | | | |

| | | | | | | | DK NF | | | | | | | |

| | | | | | | | relations | | | | | | | |

| | | | | | | +-----------+ | | | | | | |

| | | | | | +-----------------+ | | | | | |

| | | | | +----------------------+ | | | | |

| | | +----------------------------------+ | | |

| | +----------------------------------------+ | |

| +-----------------------------------------------+ |

+------------------------------------------------------+

o Higher normal forms are more restrictive.

o A relation is in a higher normal form implies that it is in a lower normal form, but *not vice versa*.

Examples 1

o If a relation is in BCNF, then it is also in 3NF, 2NF and 1NF

o If a relation is in 2NF, then

(a) it is in 1NF, and

(b) it may or may not be in 3NF, and

(c) it may or may not in BCNF.

o In general, the higher the normal forms a relation is in, the better the design of the relation (in avoiding redundancy and inconsistency).

o However, it may be necessary to consider other issues.

o 1NF and 2NF are more interesting for *historical* reasons.

o 4NF and 5NF involves the concept of *multivalued* and *join* dependencies (MVD and JD). They are hard to understand and even harder to follow in most situations.

o Domain Key Normal Form (DKNF) involves the concept of constraints.

o Based on the concept of *functional dependencies* (FD), the most important normal forms are

o 3NF and

o BCNF (*Boyce-Codd Normal Form*).

**Types Of Relationships Between *Attributes***

(1) Many to many relationships

Example: Consider the relation Enrol:

Course Student Grade

------------------------------------

C1 S1 A

C1 S2 B

C1 S3 B

C2 S1 A

C2 S4 B

Under *reasonable* assumptions, there are many to many relationships between the *attributes*

(a) Course and Student

(A course may enrol many students; a student may take many courses)

(b) Course and Grade,

(c) Student and Grade,

(d) {Course, Grade} and Student

(Both S2 and S3 have a grade of B in Course C1).

However, the relationship between {Course, Student} and Grade is not a many-to-many relationship if we assume that a student can only has one grade for a given course.

o A many to many relationship between two attributes means that there is *no constraint and no dependency* between the values of the attributes.

(2) Many-to-one relationship.

Example 1:

For many applications, the relationship between SS# and NAME are many to one.

SS# ------> NAME

(many) (one)

Interpretations and terminology:

(a) Many different SS#'s (persons) may have the same NAME.

(b) Given a SS#, there can only be one NAME associated with it (not allowing alias, etc).

(c) There should not be two tuples with the same SS#, but different NAME. For example,

SS# NAME PHONE

---------------------------------

**123456789 Peter** 367-9890

**123456789 Paul** 464-9089

222229999 Mary 787-9900

is not allowed.

(d) SS# uniquely *determines* NAME.

(e) NAME is *functionally determined* by SS#.

(f) There is a *functional dependency* SS# --> NAME.

(g) Hence, a functional dependency specifies a many to one relationship between attributes.

Example 2:

In a university, there is a many-to-one relationship between {COURSE#, STUDENT#} and GRADE.

Interpretations:

(a) A student may have only one grade for a course.

(b) We say there is a functional dependency:

COURSE# STUDENT# --> GRADE,

{COURSE#, STUDENT#} *determines* GRADE.

(c) Note that under different assumptions, the functional dependency may not be true.

(d) For example, if a student is allowed to retake a course, then he may have two grades for the same course (in different semesters), then COURSE# STUDENT# --> GRADE is *false*.

We actually have

COURSE# STUDENT# SEMESTER --> GRADE

o Hence, *functional dependency is a result of analyzing the applications*. There is no universally true *non-trivial* functional dependency.

o In other words, functional dependencies depend on the *semantic* of the problems.

Example 1:

In most application, we have

SS# --> NAME (i.e. a person has only one SS#.)

However, in a criminal database, several bad guys may use the same fake SS#, and thus

SS# --> NAME is not true.

Or, if you are dealing with an international data base with many countries. Each country may has its own SS#. Two countries may issue the same SS#. Hence,

SS# --> NAME is not true.

We may instead have SS# COUNTRY --> NAME.

**Formal definition of functional dependency**

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

i.e. the same value of X implies the same value of Y.

Example 1: SS# --> SNAME:

There are no two tuples with the same SS# but different name.

Example 2: DEPT-NO --> MANAGER-NO:

There are no two tuples with the same DEPT-NO but different MANAGER-NO. A department has only one manager.

Example 3: SUPPLIER# PART# DATE --> QUANTITY

There are no two tuples with the same SUPPLIER#, PART# *and* DATE but different QUANTITY. That is, any supplier has only one shipment of a part in a given date.

**Armstrong's Axiom**

(1) *Reflexivity* If X and Y are sets of attributes and Y is a subset of X, then X --> Y.

Example 1: Let X be CITY STREET, Y be STREET, then

Y is a subset of X, and X --> Y or

CITY STREET --> STREET.

Interpretation: If two tuples have the same value of CITY STREET, then they have the same value of STREET.

o This is so trivial that we call a functional dependency likes CITY STREET --> STREET a *trivial functional dependency*.

Example 2: A --> A and B C --> B are trivial.

o Since trivial functional dependencies do not actually give you any information, we are only interested in *non-trivial* functional dependency.

(2) *Augmentation* If X --> Y then X Z --> Y Z

Example: If S# --> SNAME, then S# P# --> SNAME P#.

(3) *Transitivity* If X --> Y and Y --> Z then X --> Z

Example: If EMP-NO --> DEPT-NO

DEPT-NO --> MANAGER-NO

then EMP-NO --> MANAGER-NO.

Interpretation: If

(a) every employee works for only one department, and

(b) every department has only one manager, then

(c) every employee has only one manager.

(4) *Decomposition* Rule: If X --> Y Z, then X --> Y and X --> Z.

(5) *Union* Rule: If X --> Y and X --> Z then X --> Y Z.

**Keys and Superkeys**

o We can use functional dependencies to define keys and superkeys.

o For a relation scheme R, K is a candidate key if

(1) Uniqueness: K --> R.

(2) Minimality: there is no proper subset of K that determines R.

o K is a superkey if K --> R.

Example:

In EMPLOYEE(EMP-NO, DEPT-NO, MANAGER-NO) with

EMP-NO --> DEPT-NO and

DEPT-NO --> MANAGER-NO.

By transitivity EMP-NO --> MANAGER-NO.

By rule union, EMP-NO --> EMP-NO DEPT-NO MANAGER-NO.

Hence, EMP-NO is a key of R(EMP-NO, DEPT-NO, MANAGER-NO).

On the other hand, DEPT-NO is not a key since we do not have DEPT-NO --> EMP-NO.