FUNCTIONAL DEPENDENCY

R(A_1, A_2, ..., A_n) : a relation SCHEMA

$X \subseteq R$, and $Y \subseteq R$: attributes in R

$r(R)$: a specific relation of type R

- R satisfies the *functional dependency* (fd) X → Y

  if *each* specific relation (relational VALUE) r(R) satisfies X → Y.

- A relation value r satisfies X → Y

  if each X-value in r is associated with a unique Y-value in r.

  In other words, a relation value r satisfies X → Y if

  for any two tuples t_1 and t_2 in r,

$$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y].$$

Consider the relation schema SP′(S#, P#, QTY, STATUS).

Relation value r(SP′) for the relation schema SP′

S#	P#	QTY	STATUS
s1	p1	100	30
s1	p2	200	30
s2	p1	100	50
s2	p3	100	50

Which functional dependencies are satisfied by this relation value?

- Full functional dependency VS. partial functional dependency

  Attribute Y is *fully* functionally dependent on attribute X, if $X \rightarrow Y$, but we do NOT have $X' \rightarrow Y$ for any proper subset X' of X.

  Consider relation R(MOTHER, CHILD, GIFT, DATE), with functional dependencies

  $F = \{CHILD \rightarrow MOTHER, (CHILD, DATE) \rightarrow GIFT\}$.

  is the attribute MOTHER fully functionally dependent on (CHILD, DATE)?

  The answer is: NO.
  Because $CHILD \rightarrow MOTHER$, and thus MOTHER is dependent on a proper subset of (CHILD, DATE).

  Attribute MOTHER is partially dependent on (CHILD, DATE).

  Consider the following sample data (relation value) for the relation R:

MOTHER	CHILD	GIFT	DATE
Linda	Jim	Bike	1-20-2009
Linda	Jim	Sweater	12-25-2008
Linda	Jim	CD	3-8-2007
Linda	Mary	Game	12-25-2008
Linda	Mary	Book	2-20-2006
Susan	David	CD	7-20-2009
Susan	David	Shoes	8-30-2009

  Do you see any problems (redundancy, etc.) from the above sample data?

EXAMPLE OF FUNCTIONAL DEPENDENCY

The two most important things to remember about *functional dependency* (fd) are:

(1) Fd's are determined by the *meaning* of the attributes and their role in the "real world" which is being modeled by the database.

(2) Fd's are in turn used to group the attributes together to form the normalized relations of the database.

Example. Consider the attributes

C	Class (of a course)
T	Time for the class
R	Room for the class
I	Instructor of the class

which describe the arrangement of room and time for classes taught by the instructors. These attributes are used to model part of the "real world". We have certain constraints about the objects in the "real world"s:

(1) No two instructors teach the same course.

(2) At any time and in a given room, there is at most one class being taught there.

(3) No class can be taught at one given time in two rooms.

(4) No instructor can teach two classes at one given time.

The functional dependencies are:

(a) $C \rightarrow I$
(b) $TR \rightarrow C$
(c) $CT \rightarrow R$
(d) $IT \rightarrow C$

Given below is a database which has some tuples violating the above functional dependencies.

Class	Instructor	Time	Room
Csc4402	Chen	1:30/M	101
His7700	Brown	3:00/T	205
Csc4402	Smith	2:00/F	310 (a)
Art2450	John	3:00/T	205 (b)
His7700	Brown	3:00/T	208 (c)
Csc7420	Chen	1:30/M	202 (d)

It is easy to see that the tuples marked (a), (b), (c), (d) violate the functional dependencies (a), (b), (c), (d) respectively.

AXIOMS FOR FUNCTIONAL DEPENDENCIES

Logical implications of dependencies. Let R(ABC) be a relation schema and F = {A $\rightarrow$ B, B $\rightarrow$ C} be the set of functional dependencies that hold on R. It is easy to see that the fd A $\rightarrow$ C also holds in R.

In general, let F be a set of functional dependencies and let X $\rightarrow$ Y be a functional dependency. We say F *logically implies* X $\rightarrow$ Y, written F |= X $\rightarrow$ Y, if every relation instance *r* that satisfies F also satisfies X $\rightarrow$ Y.

The closure of F

The closure of F, denoted as F^+, is defined as the set of all the functional dependencies that are logically implied by F. That is, F^+ ={X $\rightarrow$ Y | F |= X $\rightarrow$ Y}.

ARMSTRONG'S RULES

In the example relation R(ABC) given earlier, we noticed that from the following functional dependencies {A $\rightarrow$ B , B $\rightarrow$ C}, we can deduce A $\rightarrow$ C by transitivity, and deduce AB $\rightarrow$ C by a simple application of the definition of *functional dependency*. In fact, using the following Armstrong's rules, we can deduce AB $\rightarrow$ C from either A $\rightarrow$ C or B $\rightarrow$ C.

Armstrong's rules:

(1)	$X \rightarrow X'$ for all subsets $X' \subseteq X$	(projection rule)
(2)	if $X \rightarrow Y$ then $XZ \rightarrow YZ$	(augmentation rule)
(3)	if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$	(transitive rule)

We can also derive many other rules by repeated application of the rules (1) - (3). For example, we can obtain the *union* rule:

if $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$.

Also, we can derive the *decomposition rule*:

if $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$.

The importance of the Armstrong's rules lies in that they allow manipulation of the fd's purely in a syntactic way (at the schema level) without looking at the tuples of a relation (at the relation instance level).

DETERMINING DEPENDENT ATTRIBUTES

Let X be a set of attributes and F be a set of functional dependencies. Let X^+ be the set of all attributes that depend on a subset of X with respect to F, i.e., X^+ is the set of attributes Z, such that $X \rightarrow Z \in F^+$. X^+ is called the *closure* of X (w.r.t. F). By definition, $X \subseteq X^+$.

The algorithm for computing X^+ (w.r.t. F):

Input: A relation schema R, a set of functional dependencies F and a set of attributes X.
Output: X^+.

```
(1)   X+ ← X.
(2)   Repeat
          found ← false;
          for (each fd Y → Z ∈ F) do
           if (Y ⊆ X+)
           then begin
                found ← true;
                X+ ← X+ ∪ Z;
                remove Y → Z from further consideration;
                end;
      until (found = false) or (X+ = all attributes).
```

Example. The computation of $(AC)^+$ = ABCDE w.r.t. F = {AB $\rightarrow$ E, AD $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D} is shown below.

Fd's	Value of $(AC)^+$ after each iteration of repeat-loop		
	#1	#2	#3
AB $\rightarrow$ E (Y = AB)	–	–	ABCDE
AD $\rightarrow$ B (Y = AD)	–	ABCD	*
B $\rightarrow$ C (Y = B)	–	ABCD	*
C $\rightarrow$ D (Y = C)	ACD	*	
found	true	true	true

"*" indicates that the fd will not be considered further

Note that the order in which the fd's are used affects the details of the progress of computation, but not the final X^+.

Example. If F = {AB $\rightarrow$ C, C $\rightarrow$ B, C $\rightarrow$ A}, then we can use the following functional dependencies as a shorthand to represent the F^+: (The right hand side of each fd is simply the X^+ where X = the left hand side; the X's are all the non-empty subsets of the attributes (ABC)).

A	$\rightarrow$	A
B	$\rightarrow$	B
C	$\rightarrow$	ABC
AB	$\rightarrow$	ABC
AC	$\rightarrow$	ABC
BC	$\rightarrow$	ABC
ABC	$\rightarrow$	ABC

EQUIVALENCE AND REDUNDANCY OF FD'S

Two sets of fd's F_1 and F_2 are said to be *equivalent*, $F_1 \equiv F_2$, if they have the same closure:

$$F_1^+ = F_2^+,$$

or equivalently,

$$F_1 \subseteq F_2^+ \text{ and } F_2 \subseteq F_1^+.$$

Note that to test $F_1 \subseteq F_2^+$, we need not compute all of F_2^+. We need only to verify that for each $X \rightarrow Y \in F_1$, we have $X \rightarrow Y \in F_2^+$, i.e., the closure X^+ computed using F_2 contains Y.

Example. Let $F_1 = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ and $F_2 = \{A \rightarrow C, C \rightarrow B, B \rightarrow A\}$. Then $F_1 \equiv F_2$.

This is true because A^+ using F_2 equals ABC which contains B. Thus $A \rightarrow B \in F_2^+$. Similarly, $B \rightarrow C$ and $C \rightarrow A$ are in F_2^+, etc.

Redundant fd. An fd $f \in F$ is redundant if $F \equiv F - \{f\}$.

Theorem. $F_1 \equiv F_2$ if and only if for each subset of attributes X, $X_1^+ = X_2^+$, where X_i^+ is the closure of X computed using the fd's in F_i.

Theorem. An fd $X \rightarrow Y$ in F is redundant if and only if $Y \subseteq X^+$, where X^+ is computed without using the fd $X \rightarrow Y$.

A set of fd's is said to be *reduced* if it contains no redundant fd's.

Example. The set $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A, A \rightarrow C, C \rightarrow B, B \rightarrow A\}$ can be reduced in two ways by successively

removing the redundant fd's.

1. $\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

2. $\{A \rightarrow C, C \rightarrow B, B \rightarrow A\}$

Note that removing a redundant fd from a set F does not affect the closure F^+. There are some other ways to reduce the set F, you may try to fi nd them.

MINIMAL COVERS

Let F_1 and F_2 be two sets of functional dependencies. If $F_1 \equiv F_2$, then we say the F_1 is a cover of F_2 and F_2 is a cover of F_1. We also say that F_1 covers F_2 and vice versa. It is easy to show that every set of functional dependencies F is covered by a set of functional dependencies G, in which the right hand side of each fd has only one attribute.

We say a set of dependencies F is *minimal* if:

(1) Every right hand side of each fd in F is a single attribute.

(2) The left hand side of each fd does not have any redundant attribute, i.e., for every fd $X \rightarrow A$ in F where X is a composite attribute, and for any proper subset Z of X, the functional dependency $Z \rightarrow A \notin F^+$.

(3) F is reduced (without redundant fd's). This means that for every $X \rightarrow A$ in F, the set $F - \{X \rightarrow A\}$ is NOT equivalent to F.

Minimal Covers of F. It is easy to see that for each set F of functional dependencies, there exists a set of functional dependencies F′ such that $F \equiv F'$ and F′ is minimal. We call such F′ a *minimal cover* of F.

The algorithm to compute F′, a minimal cover of F.

Input: F, a set of fd's.
output: F′, a minimal cover of F.

(1) Let F′ = {X → A | X → A ∈ F and A is a single attribute}. For each fd X → $A_1 A_2$ ... A_n ∈ F (n > 1), put the fd's X → A_1, X → A_2, ..., X → A_n into F′, where A_i is a single attribute.

(2) **While**
there is an fd X → A ∈ F′ such that X is a composite attribute and Z ⊂ X is a proper subset of X and Z → A ∈ $(F')^+$,
do
replace X → A with Z → A.

(3) For each fd X → A ∈ F′, check if it is redundant, if it is, eliminate it. ♣

It is important to note that for the above algorithm, the ordering between step (2) and step (3) is critical: If you first perform step (3) and then perform step (2) of the algorithm, the resulting set of fd's may still have redundant functional dependencies.

It should be pointed out that for a set of functional dependencies F, there may be more than one minimal covers of F.

Example. (Computing a minimal cover.)
Let R = R(ABCDEGH) and F = {CD $\rightarrow$ AB, C $\rightarrow$ D, D $\rightarrow$ EH, AE $\rightarrow$ C, A $\rightarrow$ C, B $\rightarrow$ D}. The process of computing a minimal cover of F is as follows:

(1) **Break down the right hand side of each fd's.** After performing step (1) in the algorithm, we get F′ = {CD $\rightarrow$ A, CD $\rightarrow$ B, C $\rightarrow$ D, D $\rightarrow$ E, D $\rightarrow$ H, AE $\rightarrow$ C, A $\rightarrow$ C, B $\rightarrow$ D}.

(2) **Eliminate redundancy in the left hand side.** The fd CD $\rightarrow$ A is replaced by C $\rightarrow$ A. This is because C $\rightarrow$ D $\in (F')^+$, hence C $\rightarrow$ CD $\in (F')^+$; from C $\rightarrow$ CD $\in (F')^+$ and CD $\rightarrow$ A $\in$ F′, by transitivity, we have C $\rightarrow$ A $\in (F')^+$ and hence CD $\rightarrow$ A should be replaced by C $\rightarrow$ A. Similarly, CD $\rightarrow$ B is replaced by C $\rightarrow$ B, AE $\rightarrow$ C is replaced by A $\rightarrow$ C. F′ = {C $\rightarrow$ A, C $\rightarrow$ B, C $\rightarrow$ D, D $\rightarrow$ E, D $\rightarrow$ H, A $\rightarrow$ C, B $\rightarrow$ D} after step (2).

(3) **Remove redundant fd's.** The fd C $\rightarrow$ D is eliminated because it can be derived from C $\rightarrow$ B and B $\rightarrow$ D and hence it is redundant. The F′ now becomes {C $\rightarrow$ A, C $\rightarrow$ B, D $\rightarrow$ E, D $\rightarrow$ H, A $\rightarrow$ C, B $\rightarrow$ D}, which is the only minimal cover of F. ♣

CANDIDATE KEYS

A *candidate key* of a relation schema R is a subset X of the attributes of R with the following two properties:

1. Every attribute is functionally dependent on X,
   i.e., X^+ = all attributes of R (also denoted as X^+ = R).

2. No proper subset of X has the property (1),
   i.e., X is minimal with respect to the property (1).

A *sub-key* of R: a subset of a candidate key;

a *super-key*: a set of attributes containing a candidate key.

We also use the abbreviation CK to denote "candidate key".

Let R(ABCDE) be a relation schema and consider the following functional dependencies F = {AB $\rightarrow$ E, AD $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D}. Since

$(AC)^+$ =ABCDE,
A^+ = A, and
C^+ = CD,

we know that AC is a candidate key, both A and C are sub-keys, and ABC is a super-key. The only other candidate keys are AB and AD. Note that since nothing determines A, A is in every candidate key.