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.

Computing All Candidate Keys of a Relation R.

Given a relation schema R(A_1, A_2, ..., A_n) and a set of functional dependencies F which hold true on R, how can we compute all candidate keys of R?

Since each candidate key must be a minimal subset Z of $\{A_1, ..., A_n\}$ such that Z^+ = R, we have the following straightforward (and brute-force) algorithm:

(1) Construct a list L consisting of all non-empty subsets of $\{A_1, ..., A_n\}$ (there are $2^n - 1$ of them). These subsets are arranged in L in ascending order of the size of the subset: We get L = $\langle Z_1, Z_2, ..., Z_{2^n-1}\rangle$, such that $|Z_i| \leq |Z_{i+1}|$. Here $|Z_i|$ denotes the number of elements in the set Z_i.

(2) Initialize the set K = {} (K will contain all CK's of R).
While L is not empty, remove the first element Z_i from L, and compute Z_i^+.
If Z_i^+ = R, then

(a) Add Z_i to K

(b) Remove any element Z_j from L if $Z_i \subset Z_j$ (because Z_j is too big, it can not be a CK).

(3) Output K as the final result.

ANALYSIS The method in previous page is correct but not efficient: the list L is too big (exponential in the number of attributes in R).

IDEA:

- Focus on "necessary" attributes that will definitely appear in ANY candidate key of R.
- Ignore "useless" attribute that will NEVER be part of a candidate key.

Necessary attributes:
An attribute A is said to be a necessary attribute if

(a) A occurs only in the L.H.S. (left hand side) of the fd's in F; or

(b) A is an attribute in relation R, but A does not occur in either L.H.S. or R.H.S. of any fd in F.

In other words, necessary attributes NEVER occur in the R.H.S. of any fd in F.

Useless attributes:
An attribute A is a useless attribute if A occurs ONLY in the R.H.S. of fd's in F.

Middle-ground attributes:
An attribute A in relation R is a middle-ground attribute if A is neither necessary nor useless.

Example.
Consider the relation R(ABCDEG) with set of fd's F = {AB $\rightarrow$ C, C $\rightarrow$ D, AD $\rightarrow$ E}

Necessary	Useless	Middle-ground
A, B, G	E	C, D

An important observation about necessary attribute is: a necessary attribute will appear in *every* CK of R, and thus ALL necessary attributes must appear in every CK of R.

If X is the collection of ALL necessary attributes and X^+ = All attributes of R, then X must be the ONLY candidate key of R. (Think: Why is that?)

Thus we should first check the necessary attribute closure X^+, and terminate the CK computing algorithm when X^+ = R. On the other hand, we also notice that useless attributes can never be part of any CK. Therefore, in case $X^+ \neq$ R, and we have to enumerate subsets Z_i of R to find CK's, the Z_i's should NOT contain any useless attributes, and each Z_i should contain all the necessary attributes. In fact, the list L of subsets to test should be constructed from all non-empty subsets of the middle-ground attributes, with each subset expanded to include all necessary attributes.

The algorithm for computing all candidate keys of R.

Input: A relation R={ A_1, A_2, ..., A_n}, and F, a set of functional dependencies.
Output: K={ $K_1, \ldots, K_t$}, the set of all candidate keys of R.

Step1.

Set F′ to a minimal cover of F (This is needed because otherwise we may not detect all useless attributes).

Step2.

Partition all attributes in R into necessary, useless and middle-ground attribute sets according to F′. Let X={ $C_1, \ldots, C_l$} be the necessary attribute set, Y = {B_1, ..., B_k} be the useless attribute set, and M = {A_1, ..., A_n} – (X ∪ Y) be the middle-ground attribute set.
If X={}, then go to step4.

Step3.

Compute X^+. If X^+ =R, then set K= {X}, terminate.

Step4.

Let L = $\langle Z_1, Z_2, \ldots, Z_m \rangle$ be the list of all non-empty subsets of M (the middle-ground attributes) such that L is arranged in ascending order of the size of Z_i.
Add all attributes in X (necessary attributes) to each Z_i in L.

Set K = {}.
i ← 0.
WHILE L ≠ empty do
BEGIN
i ← i + 1.
Remove the first element Z from L.
Compute Z^+.
If Z^+ = R,
then
begin
set K ← K ∪ {Z};
for any Z_j ∈ L, if $Z \subset Z_j$
then L ← L − {Z_j}.
end
END

♣

Example. (Computing all candidate keys of R.)
Let R = R(ABCDEG) and F = {AB $\rightarrow$ CD, A $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ E, BD $\rightarrow$ A}. The process to compute all candidate keys of R is as follows:

(1) The minimal cover of F is {A $\rightarrow$ B, A $\rightarrow$ D, B $\rightarrow$ C, C $\rightarrow$ E, BD $\rightarrow$ A}.

(2) Since attribute G never appears in any fd's in the set of functional dependencies, G must be included in a candidate key of R. The attribute E appears only in the right hand side of fd's and hence E is not in any key of R. No attribute of R appears only in the left hand side of the set of fd's. Therefore X = G at the end of step 2.

(3) Compute G^+ = G, so G is not a candidate key.

(4) The following table shows the L, K, Z and Z^+ at the very beginning of each iteration in the **WHILE** statement.

i	Z	Z^+	L	K
0	–	–	⟨AG, BG, CG, DG, ABG, ACG, ADG, BCG, BDG, CDG, ABCG, ABDG, ACDG, BCDG, ABCDG⟩	{}
1	AG	ABCDEG = R	⟨BG, CG, DG, BCG, BDG, CDG, BCDG⟩	{AG}
2	BG	BCEG ≠ R	⟨CG, DG, BCG, BDG, CDG, BCDG⟩	{AG}

3	CG	CEG ≠ R	⟨DG, BCG, BDG, CDG, BCDG⟩	{AG}
4	DG	DG ≠ R	⟨BCG, BDG, CDG, BCDG⟩	{AG}
5	BCG	BCEG ≠ R	⟨BDG, CDG, BCDG⟩	{AG}
6	BDG	ABCDEG = R	⟨CDG⟩	{AG, BDG}
7	CDG	CEDG ≠ R	⟨ ⟩	{AG, BDG}

GOALS OF NORMALIZATION

- Let R(A_1, ..., A_n) be a relation schema with a set F of functional dependencies.

- Decide whether a relation scheme R is in "good" form.

- In the case that a relation scheme R is not in "good" form,

  decompose it into a set of smaller relation schemas $\{R_1, R_2, ..., R_m\}$ such each relation schema R_j is in "good" form (such as 3NF or BCNF).

- Moreover, the the decomposition is a lossless-join (LLJ) decomposition

- Preferably, the decomposition should be functional dependency preserving.

LOSS-LESS-JOIN DECOMPOSITION

The decomposition

$$SP'(S\#, P\#, QTY, STATUS) = SP(S\#, P\#, QTY) \otimes S'(S\#, STATUS)$$

is called a *loss-less-join* (LLJ) decomposition, or we say that the decomposition is loss-less, because the join of the component relations SP(S#, P#, QTY) and S′(S#, STATUS) gives back the original relation SP′(S#, P#, QTY, STATUS).

In general, a decomposition of relation schema R into R_1, ..., R_m is an LLJ decomposition if the natural join of R_1, ..., R_m equals the original relation R:

$$R = R_1 \otimes R_2 \otimes \cdots \otimes R_m$$

We must be very careful in performing the decomposition otherwise the loss-less property may not be maintained.

Intuitively, the decomposition of SP′ = SP ⊗ S′ is loss-less because the attribute "S#" which is mutual to SP and S′ can uniquely determine the attribute "STATUS".

Theoretically, we have the following theorem which tells us the suffi cient condition for a decomposition to be loss-less.

Heath Theorem. For a relation R with attributes A, B, C and functional dependency A → B, the decomposition

$$R(A, B, C) = R_1(A, B) \otimes R_2(A, C)$$

is always loss-less.

Functional Dependency-Preserving Decomposition

The two most desirable properties of any decomposition of a relation R are:

(1) loss-less, and
(2) fd-preserving.

- Let R be a relation schema with functional dependencies F.

R decomposed into $R_1, R_2, ..., R_m$

F projected into $F_1, F_2, ..., F_m$

- The decomposition is LLJ:
  $R = R_1 \otimes R_2 \otimes \cdots \otimes R_m$

- The decomposition is fd-preserving:
  $F_1 \cup F_2 \cup ... \cup F_m \equiv$ F.

  Here F_j: the projection of F onto R_j, i.e., $F_j = \{X \rightarrow Y \mid X \rightarrow Y \in F^+$, and $X \subseteq R_j$, $Y \subseteq R_j\}$.

  F_j is the set of fd's from the closure of F, such that the fd's contain only attributes from R_j.

FIRST, SECOND NORMAL FORM

First normal form. A relation R is said to be in *first normal form* if all the underlying domains of R contain only atomic values. We write 1NF as a short hand for first normal form.

Non-key attribute. An attribute A in relation R is said to be a non-key attribute if it is not a subkey, i.e., A is not a component of any candidate key K of R.

Second normal form. A relation R is said to be in *second normal form* if R is in 1NF and every non-key attribute is *fully* dependent on each candidate key K of R. In Example 1, the relation SP′ has only one candidate key (S#, P#). The attribute STATUS is a non-key attribute of SP′ and it is not fully dependent on (S#, P#), therefore SP′ is not in 2NF.

Although relations in 2NF have better properties than those in 1NF, there are still problems with them.

These problems motivate the search for more "ideal" form of relations and hence the 3NF and BCNF normal forms are identified.

THIRD NORMAL FORM AND 3NF DECOMPOSITION

A relation schema *R* with a set fd's F is said to be in third normal form (3NF) if for each fd $X \rightarrow A \in F$ (here A is a single attribute), we have

either X is a superkey

or A is a prime.

A single attribute A is called a prime if A is a subkey, i.e., A is a component of a key.

Example. The relation (ADB) with $F = \{AD \rightarrow B, B \rightarrow D\}$ is in 3NF, because AD and AB are its candidate keys - so every attribute is a prime. Thus the fd $B \rightarrow D$ does not cause a violation of the 3NF condition: although B is not a superkey, D is a prime.

Bernstein's Theorem. Every relation *R* has a loss-less, fd-preserving 3NF decomposition.

BERNSTEIN'S ALGORITHM

Input: A relation schema R with given attributes and a set F of functional dependencies that hold on R such that F is already a minimal cover of itself.

Output: A set of 3NF relations that form a loss-less and fd-preserving decomposition of R.

Algorithm:

1. Group together all fd's which have the same L.H.S. If $X \rightarrow Y_1$, $X \rightarrow Y_2$, ..., $X \rightarrow Y_k$ are all the fd's with the SAME L.H.S. X, then replace all of them by the single fd $X \rightarrow Y_1 Y_2 \ldots Y_k$.

2. For each fd $X \rightarrow Y$ in F, form the relation (XY) in the decomposition.

3. IF $X'Y' \subset XY$, then remove the relation $(X'Y')$ from the decomposition.

4. If none of the relations obtained after step (3) contains a candidate key of the **original relation R**, then find a candidate key K of R and add the relation (K) to the decomposition.

Example. This example shows the need for steps 1, 3 and 4. Given R(ABCDE) and F = {A $\rightarrow$ B, A $\rightarrow$ C, C $\rightarrow$ A, BD $\rightarrow$ E}.

Step 1. {A $\rightarrow$ BC, C $\rightarrow$ A, BD $\rightarrow$ E}.

Step 2. R_1(ABC), R_2(CA), R_3(BDE)

Step 3. R_1(ABC), R_3(BDE)

Step 4. R_1(ABC), R_3(BDE) and R_4(AD)

So the final decomposition is:
R(ABCDE) = R_1(ABC) $\otimes$ R_3(BDE) $\otimes$ R_4(AD)

Note that we add the relation R_4(AD) to the list of final decomposition in step 4 of the algorithm, b/c the relation R has two CK's: AD and CD, and neither is contained in any of the R_1(ABC) and R_3(BDE). So we need to add either (AD) or (CD) (not both, though) to the finmal list of relations.

BOYCE-CODD NORMAL FORM (BCNF)

- A relation schema *R* is in BCNF, if for every **nontrivial** functional dependency X $\rightarrow$ Y that holds on *R*, the attribute X is a superkey. An fd X $\rightarrow$ Y is **trivial**, if Y is a subset of X, i.e., Y $\subseteq$ X.

- Given a relation schema R and a set F of functional dependencies that hold on R, how can we tell whether R is in BCNF or not?
  We only need to check, for each fd X $\rightarrow$ Y in F, the L.H.S. attribute X is a super key or not, i.e., whether X^+ = R.

Consider the relation schema R(ABCDE) and the fd's F = {AB $\rightarrow$ E, AD $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D}. Relation R has 3 candidate keys AB, AC, AD. Obviously R is not in BCNF, because we have the fd's B $\rightarrow$ C and C $\rightarrow$ D in R, but B is not a superkey of R, neither is C. By using the Heath Theorem, we can obtain an LLJ decomposition of R into three BCNF relations as follows:

R(ABCDE)

by C $\rightarrow$ D

R_1(CD) R′(ABCE)

by B $\rightarrow$ C

R_2(BC) R_3(ABE)

The decomposition is given by R(ABCDE) = R_1(CD) $\otimes$ R_2(BC) $\otimes$ R_3(ABE).

Algorithm to decompose a relation R into a set of BCNF relations

Input: Relation R and set of functional dependencies F (F is minimal)
Output: Result={ R_1, R_2, ..., R_m } such that
(A) Each R_i is in BCNF
(B) The decomposition is loss-less-join decomposition

1. Group together all fd's which have the same L.H.S. If $X \rightarrow Y_1$, $X \rightarrow Y_2$, $\cdots$, $X \rightarrow Y_k$ are in F, then replace all of them by a single fd $X \rightarrow Y_1\ Y_2 \cdots Y_k$.
result := {R};
done := FALSE;
Compute F^+;

2. **WHILE** (**NOT** done) **DO**
IF (there is a schema R_i in *result* that is not in BCNF)
THEN BEGIN
Let X→Y be a nontrivial functional dependency in F^+ that holds on R_i
such that X→ R_i is not in F^+;
result :=(*result* − {R_i}) ∪ {(R_i − Y), (XY)};
(this means to break R_i into two relations R_i − Y and XY)
END
ELSE done := **TRUE** ;

Example. (LLJ decomposition of R into a set of BCNF relations.) Let R = R(ABCDEGH) and F = {B $\rightarrow$ E, B $\rightarrow$ H, E $\rightarrow$ A, E $\rightarrow$ D, AH $\rightarrow$ C}. The decomposition according to the given algorithm is as follows:

(1) After grouping the R.H.S of the fd's together, we get

$$\{B \rightarrow EH, E \rightarrow AD, AH \rightarrow C\}.$$

(2) The only candidate key of R is BG. We have the following LLJ decomposition:

R(ABCDEHG)

by AH $\rightarrow$ C

R_1(AHC) R′(ABDEHG)

by E $\rightarrow$ AD

R_2(EAD) R″(BEHG)

by B $\rightarrow$ EH

R_3(BEH) R_4(BG)

It is easy to verify that the four resulting relations are all in BCNF, and

$$R(ABCDEHG) = (AHC) \otimes (EAD) \otimes (BEH) \otimes (BG).$$

Also we can see that all the functional dependencies in F are preserved.

PROBLEM WITH BCNF DECOMPOSITION

A problem with BCNF is that we may not be able to find a BCNF decomposition of a relation R such that the decomposition has the properties of being both loss-less and fd-preserving.

We know that we can always find an LLJ decomposition of a relation R into a set of BCNF relations. However, we may not find a BCNF decomposition of a relation R such that the decomposition is both LLJ AND fd-preserving.

For example, for relation schema R(ABD) with functional dependencies F = {AD $\rightarrow$ B, B $\rightarrow$ D}, R is NOT in BCNF, b/c in the fd B $\rightarrow$ D, attribute B is NOT a super key of R. So we use Heath theorem to get the decomposition R(ADB) = R_1(BD) $\otimes$ R_2(BA). However, the fd AD $\rightarrow$ B is not preserved by this decomposition. One cannot verify AD $\rightarrow$ B be looking at the tuples of either of the R_1(BD) or R_2(BA). This means that we must reconstruct the original relation R(ADB) to verify AD $\rightarrow$ B. This is, however, undesirable.