Functional Dependencies and Database Normalization

Finding all candidate keys for a relation with a given set of functional dependencies (FDs):

Candidate Key: Minimal superkey. Has two properties:

1. Its attribute closure (the set of attributes whose values can be determined using the attributes in the candidate key) is equal to the list of all attributes in the relation.
2. It is minimal (i.e. if we remove any attribute from the candidate key, the remaining attributes **do not** form a candidate key anymore)

Algorithm to find all candidate keys for a relation:

Step 1: Find the list of attributes that appear only on the left-hand side of the FDs
Step 2: Find the list of attributes that appear only on the right-hand side of the FDs
Step 3: Find the list of attributes that do not appear in the FDs at all
*Step 4: Find the union of the attribute sets in Step 1 and Step 3 →These are the attributes that **must** be part of all candidate keys*
*Step 5: Calculate the attribute closure $\{X\}^+$ of the attribute set {X} found in Step 4. If the closure includes all attributes of the relation, then {X} is a candidate key, and it is the **only** candidate key. If not, add one attribute (from the list of attributes **not** among those found in Step 2) at a time to the set and update the attribute closure based on the FDs until the attribute closure contains all attributes in the relation (Never add attributes which were already identified as candidate keys, as candidate keys have to be minimal).*

Example:

For the relation R with the functional dependency set below, find the highest normal form of R.

R (A, B, C, D, E)

A → BC
CD → E
B → D
E → A

Find the candidate keys of R.

Step 1: only on the left: -
Step 2: only on the right: -
Step 3: not appearing in the FDs: -
Step 4: we start with the empty set for the candidate keys.

Step 5:

1. Adding A: $\{A\}^+$: From A→BC, we get BC in the closure
From B→ D, we get D in the closure
From CD→ E, we get E in the closure
=> $\{A\}^+$=ABCDE, hence **A** is a candidate key

2. Adding B: $\{B\}^+$: From B→ D, we get D in the closure => $\{B\}^+$=BD

3. Adding C: $\{C\}^+$: No FDs => $\{C\}^+$=C

4. Adding D: $\{D\}^+$: No FDs => $\{D\}^+$=D

5. Adding E: $\{E\}^+$: From E→A, we get A in the closure
From A→ BC, we get BC in the closure
From B→D, we get D in the closure
=> $\{E\}^+$=ABCDE, hence **E** is a candidate key

2.1. Adding C to {B}: $\{BCD\}^+$: From CD→ E, we get E in the closure
From E→ A, we get A in the closure
=> $\{BC\}^+$=ABCDE, hence **BC** is a candidate key

3.1. Adding D to {C}: $\{CD\}^+$: From CD→ E, we get E in the closure
From E→ A, we get A in the closure
From A→ BC, we get B in the closure
=> $\{CD\}^+$=ABCDE, hence **CD** is a candidate key

4.1. Nothing to add here, as E is a candidate key itself.

The candidate keys are: A, E, BC, CD.

Normal Forms:

Definitions:

Prime attribute: An attribute that is part of at least one candidate key.

Non-prime attribute: An attribute that is not part of any candidate key.

1st Normal Form (1NF): Automatically satisfied in the **flat** relational model. Every attribute has to be atomic, i.e. no composite or multi-valued attributes can be present in the relation.

2nd Normal Form (2NF): No partial dependencies are allowed. For every functional dependency X→ A, the following condition has to hold:

If A is a non-prime attribute, then X cannot be a proper subset of any candidate key.

3rd Normal Form (3NF): No transitive dependencies are allowed. For every functional dependency X→A, the following condition has to hold:

(a) X is a superkey
OR
(b) A is a prime attribute

Boyce-Codd Normal Form (BCNF): Every attribute depends on nothing but the key. For every functional dependency X→ A, the following condition has to hold:

X is a superkey

Every relation that is in BCNF is automatically in 3NF; every relation that is in 3NF is automatically in 2NF and every relation in 2NF is automatically in 1NF. In general, the more restrictive the normal form (the higher the normal form is), the better the relation. BCNF is more restrictive than 3NF, 3NF is more restrictive than 2NF and 2NF is more restrictive than 1NF.

!!! In order to find the highest normal form of a relation, you first need to find the candidate keys for that relation.

Example for finding the highest normal form of relations:

Given a relation R = {P, R, S, T, U, V, W, X, Y, Z} with the following set of functional dependencies:

$XY \rightarrow Z$
$X \rightarrow UV$
$Y \rightarrow W$
$W \rightarrow ST$
$U \rightarrow PR$

What is the highest normal form of R?

For the following decomposition of R: State whether the relations in the decomposition are in 2NF, 3NF, BCNF (given they are all in 1NF).

Decomposition = {R1, R2, R3};
Where R1 = {X, Y, Z, U, V}, R2 = {Y, W, S, T}, R3 = {U, P, R}

Solution:

We first find the candidate keys for R using the algorithm outlined above. The only candidate key is XY.

Check for 2NF:
The dependency X → UV violates 2NF, as U and V are non-prime attributes and X is a proper subset of a candidate key. Therefore, the highest normal form of R is 1NF.

For the decomposition, we need to find the candidate keys for each relation in the decomposition. Notice that we will need to revise the functional dependency set for each relation in the decomposition, as some of the dependencies may not apply due to a smaller set of attributes:

- Set of FDs that apply to R1:

XY → Z
X → UV

Given these FDs, the only candidate key for R1 is XY.

Check for 2NF: Fails due to the dependency X→UV, as X is a proper subset of the candidate key. Hence, the highest normal form of R1 is 1NF.

- Set of FDs that apply to R2:

Y→W
W→ ST

Given these FDs, the only candidate key for R2 is Y.

Check for 2NF: For the first dependency, Y is a superkey, so 2NF is not violated; for the second dependency, W is not a proper subset of the superkey, so 2NF is not violated. Therefore R2 is in 2NF.

Check for 3NF: The second dependency violates 3NF, as S and T are non-prime attributes and W is not a superkey. Hence, the highest normal form is 2NF.

- Set of FDs that apply to R3:

U → PR

Given this FD, the only candidate key for R3 is U.

This relation is in BCNF as there is only one functional dependency (where the superkey is on the left-hand side of that FD).

Database Normalization:

Database normalization is the process of decomposing relations into smaller relations to minimize redundancy and eliminate undesirable characteristics like insertion, deletion and modification anomalies (in general the new relations satisfy higher normal forms than the original relation).

Desirable properties for decomposition of relations:

1. No attributes are lost in the decomposition (**attribute preservation**), i.e. the union of the sets of attributes in the new relations is equal to the set of attributes in the original relation → This is absolutely essential.
2. All relations in the decomposition are in 3NF or BCNF.
3. The decomposition is dependency-preserving.
4. The decomposition is lossless-join (non-additive join) → This is essential.

Definitions:

Dependency-preservation: A decomposition has the dependency-preservation property if each original functional dependency either appears directly in one of the relations in the decomposition, or could be inferred from the dependencies that appear in some relation in the decomposition (i.e. dependencies are not lost). The formal definition is as follows:

Definition. Given a set of dependencies F on R, the **projection** of F on R_i, denoted by $\pi_{Ri}(F)$ where R_i is a subset of R, is the set of dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \cup Y$ are all contained in R_i. Hence, the projection of F on each relation schema R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left- and right-hand-side attributes are in R_i. We say that a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R is **dependency-preserving** with respect to F if the union of the projections of F on each R_i in D is equivalent to F; that is, $((\pi_{R_1}(F)) \cup ... \cup (\pi_{R_m}(F)))^+ = F^+$.

Example 1:
Given R (A, B, C) and F={A→B, B→C}
Is the decomposition: R1(A,C); R2(B,C) dependency-preserving?

Solution:
In R1 the following FDs hold: F1={A→C}
In R2 the following FDs hold: F2={B→C}

The FD A→B is not preserved in the decomposition, therefore the decomposition is not dependency-preserving.

Example 2:
Given R(A, B, C) and F={A→B, B→C}
Is the decomposition R1(A, B); R2(B, C) dependency-preserving?

Solution:
In R1, the following dependencies hold: F1={A→B}
In R2, the following dependencies hold: F2= {B→C}

All FDs are preserved, therefore the decomposition is dependency-preserving.

Lossless (non-additive) join: A decomposition has the lossless join property if it ensures that no spurious tuples will be generated when a natural join operation is applied to the relations resulting from the decomposition. The formal definition is as follows:

Definition. Formally, a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R has the **lossless (nonadditive) join property** with respect to the set of dependencies F on R if, for *every* relation state r of R that satisfies F, the following holds, where $*$ is the NATURAL JOIN of all the relations in D: $*(\pi_{R_1}(r), ..., \pi_{R_m}(r)) = r$.

Algorithm for testing for the lossless-join property:

Input: A universal relation R, a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R, and a set F of functional dependencies.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (* *comment* *).

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set $S(i, j) := b_{ij}$ for all matrix entries. (* each b_{ij} is a distinct symbol associated with indices (i, j) *).
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
   {if (relation R_i includes attribute A_j) then set $S(i, j) := a_j$;};}; (* each a_j is a distinct symbol associated with index (j) *).
4. Repeat the following loop until a *complete loop execution* results in no changes to S
   {for each functional dependency $X \rightarrow Y$ in F
   {for all rows in S *that have the same symbols* in the columns corresponding to attributes in X
   {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: If any of the rows has an *a* symbol for the column, set the other rows to that *same a* symbol in the column. If no *a* symbol exists for the attribute in any of the rows, choose one of the *b* symbols that appears in one of the rows for the attribute and set the other rows to that same *b* symbol in the column ;} ; } ;};
5. If a row is made up entirely of *a* symbols, then the decomposition has the nonadditive join property; otherwise, it does not.

Example for testing for the lossless-join property:

Nonadditive join test for *n*-ary decompositions. (a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test. (b) A decomposition of EMP_PROJ that has the lossless join property. (c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

(a) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2\}$

R_1 = EMP_LOCS = {Ename, Plocation}

R_2 = EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(No changes to matrix after applying functional dependencies)

(b)

EMP

Ssn	Ename

PROJECT

Pnumber	Pname	Plocation

WORKS_ON

Ssn	Pnumber	Hours

(c) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2, R_3\}$

R_1 = EMP = {Ssn, Ename}

R_2 = PROJ = {Pnumber, Pname, Plocation}

R_3 = WORKS_ON = {Ssn, Pnumber, Hours}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Original matrix S at start of algorithm)

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(Matrix S after applying the first two functional dependencies; last row is all "a" symbols so we stop)

Decomposition Algorithms

- It is always possible to find a dependency-preserving, lossless-join decomposition of a relation such that all relations in the decomposition are in 3NF.
- It is always possible to find a lossless-join decomposition of a relation into BCNF relations.
- It is **not** always possible to find a dependency-preserving, lossless-join decomposition of a relation into BCNF relations.

Algorithm for finding a dependency-preserving, lossless-join decomposition into 3NF relations:

1. Group together all FDs that have the same left-hand side (e.g. if we have the FDs X→Y1, X→Y2, X→Y3, we group them together as X→Y1Y2Y3)
2. For each FD X→Y in step 1, form the relation (XY) in the decomposition.
3. If any X'Y' is a subset of any 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, form a relation for K in the decomposition, where K is one of the candidate keys for the original relation.

Example:

Given the relation R(A, B, C, D, E) and the FDs F= {A→B, A→C, C→A, BD→E}, find a decomposition of R into 3NF relations that is lossless-join and dependency-preserving.

Solution:

Step 1: F'={A→BC, C→A, BD→E}
Step 2: R1(A, B, C); R2(A, C); R3(B, D, E)
Step 3: We remove R2, as it is a subset of R1 -> R1(A, B, C); R3(B, D, E)
Step 4: R has two candidate keys: AD and CD, none of them is contained in R1 or R3, so we form R4(A, D)

The final decomposition is: R1(A, B, C); R3(B, D, E); R4(A, D)

Algorithm for finding a lossless-join decomposition into BCNF relations:

1. Group together all FDs that have the same left-hand side (e.g. if we have the FDs X→Y1, X→Y2, X→Y3, we group them together as X→Y1Y2Y3)
2. Compute F^+: The set of all functional dependencies that can be inferred from F
3. while there are relations R_i, which are not in BCNF:
Let X→Y be a non-trivial FD in F^+ that holds on R_i such that X→R_i is not in F^+
Decompose R_i into two relations R_i-Y and XY

Example:

Given R(A, B, C, D, E, G, H) and F={B→E, B→H, E→A, E→D, AH→C}, find a lossless-join decomposition of R into BCNF relations.

Solution:

F: {AH→C, E→AD, B→EH}

The only candidate key for R is BG, hence R is not in BCNF as all FDs fail the check for BCNF.

For AH→C, AH→ABCDEGH is not in F^+, therefore we break R into R1(A, H, C) and R2(A, B, D, E, G, H)

R1 is in BCNF, but R2 is not (the only candidate key for R2 is BG and E→AD violates BCNF). For E→AD, E→ABDEGH is not in F^+, therefore we break R2 into R3(E, A, D) and R4(B, E, H, G).

R4 is not in BCNF (the only candidate key for R4 is BG and B→EH violates BCNF). For B→EH, B→BEGH is not in F^+, therefore we break R4 into R5(B, E, H) and R6(B, G).

The final decomposition is: R1(A, H, C); R3(E, A, D); R5(B, E, H); R6(B, G).

Minimal Cover of a Set of Functional Dependencies

We say that a set of functional dependencies *F* covers another set of functional dependencies *G*, if every functional dependency in G can be inferred from F.

Definition: A minimal cover of a set of functional dependencies F is a minimal set of dependencies (in the standard form and without redundancy) that is equivalent to F. We can always find at least one minimal cover for any set of functional dependencies.

Algorithm to find a minimal cover for a set of functional dependencies:

Input: A set of functional dependencies E.

1. Set $F := E$.
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$.
3. For each functional dependency $X \rightarrow A$ in F
   for each attribute B that is an element of X
   if $\{ \{F - \{X \rightarrow A\}\} \cup \{ (X - \{B\}) \rightarrow A\} \}$ is equivalent to F
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.
4. For each remaining functional dependency $X \rightarrow A$ in F
   if $\{F - \{X \rightarrow A\}\}$ is equivalent to F,
   then remove $X \rightarrow A$ from F.

Example:

Given the functional dependency set F = {B → A, D → A, AB→D}, find a minimal cover of F.

Step 1: F = {B → A, D → A, AB→D}
Step 2: All dependencies here have one attribute on the RLS, so no changes are needed.
Step 3: We need to check whether AB→D has any redundant attributes on the LHS, i.e. can we infer A→D or B→D? Using the transitivity rule with B→A and AB→D, we get B→D, therefore AB→D can be replaced with B→D. After this step, we have {B→A, D→A, B→D}.
Step 4: Using transitivity we can get the dependency B→A from the two dependencies B→D and D→A, hence B→A is redundant.

The minimal cover is {D→A, B→D}.