

Module - III (Part-2)

Normalization

3NF to 5NF



Outline

- Decomposition Using Functional Dependencies
- BCNF
- 3NF
- 4NF
- PJNF



Decomposition

- The only way to avoid the repetition-of-information problem in the *in_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose

employee(*ID*, *name*, *street*, *city*, *salary*)

into

employee1 (*ID*, *name*)

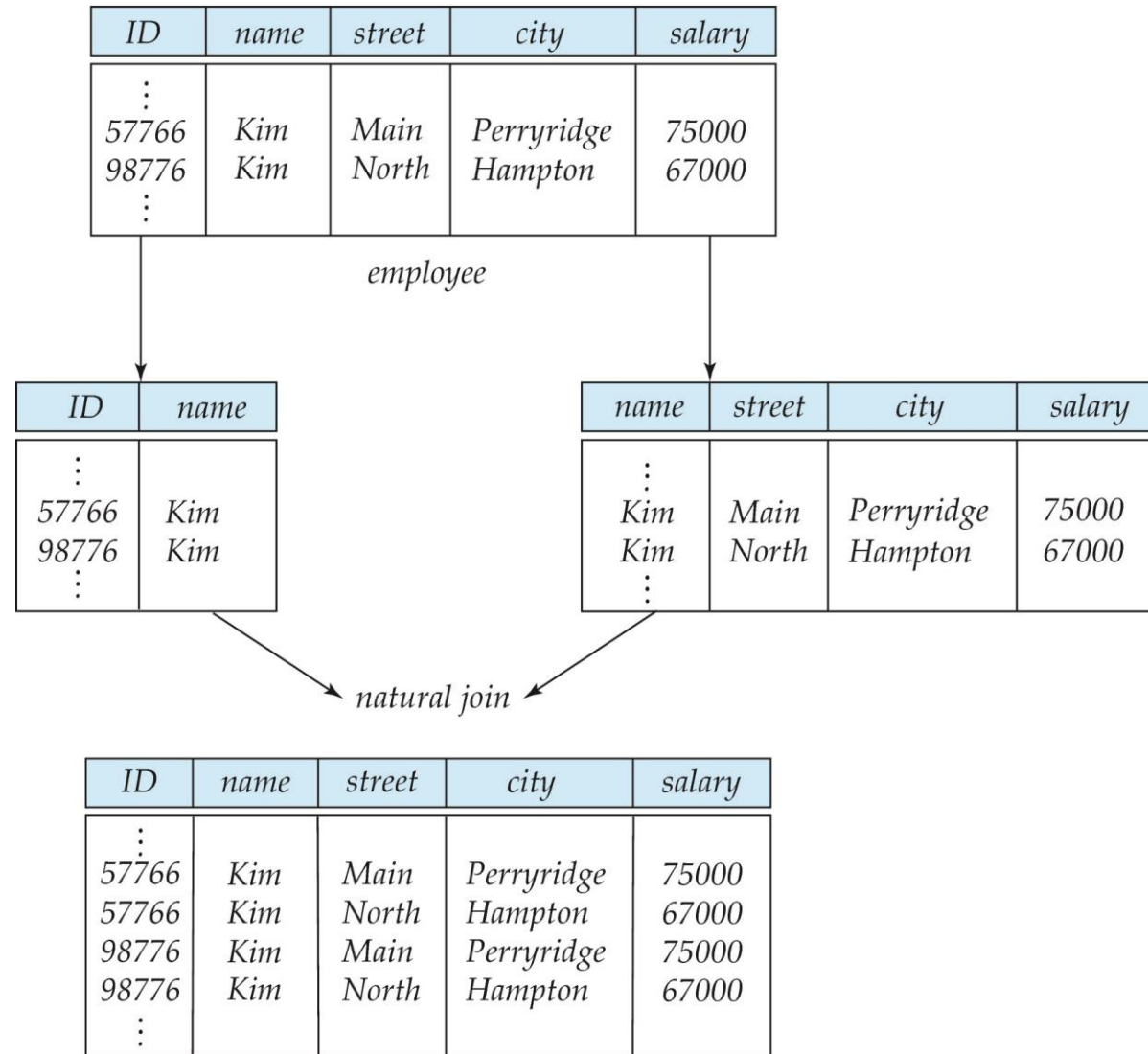
employee2 (*name*, *street*, *city*, *salary*)

The problem arises when we have two employees with the same name

- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.



A Lossy Decomposition





Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R . That is $R = R_1 \cup R_2$
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R_1 \cup R_2$
- Formally,
$$\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$
- And, conversely a decomposition is lossy if
$$r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$



Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$

$$R_1 = (A, B)$$

$$R_2 = (B, C)$$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$

A	B	C
α	1	A
β	2	B



Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless.
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+ :
 - $R_1 \cap R_2 \rightarrow R_1$
 - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies



Example

- $R = (A, B, C)$
 $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (B, C)$
 - Lossless decomposition:
 $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- $R_1 = (A, B), R_2 = (A, C)$
 - Lossless decomposition:
 $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
- *Note:*
 - $B \rightarrow BC$
is a shorthand notation for
 - $B \rightarrow \{B, C\}$



Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low
- When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Product.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT **dependency preserving**.



Dependency Preservation Example

- Consider a schema:

dept_advisor(s_ID, i_ID, department_name)

- With function dependencies:

$i_ID \rightarrow dept_name$

$s_ID, dept_name \rightarrow i_ID$

- In the above design we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.
- To fix this, we need to decompose *dept_advisor*
- Any decomposition will not include all the attributes in
 $s_ID, dept_name \rightarrow i_ID$
- Thus, the composition NOT be dependency preserving



Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R



Boyce-Codd Normal Form (Cont.)

- Example schema that is **not** in BCNF:
in_dep (ID, name, salary, dept_name, building, budget)
because :
 - *dept_name* → *building, budget*
 - holds on *in_dep*
 - but
 - *dept_name* is not a superkey
- When decompose *in_dept* into *instructor* and *department*
 - *instructor* is in BCNF
 - *department* is in BCNF



Decomposing a Schema into BCNF

- Let R be a schema R that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF.
- We decompose R into:
 - $(\alpha \cup \beta)$
 - $(R - (\beta - \alpha))$
- In our example of in_dep ,
 - $\alpha = dept_name$
 - $\beta = building, budget$and in_dep is replaced by
 - $(\alpha \cup \beta) = (dept_name, building, budget)$
 - $(R - (\beta - \alpha)) = (ID, name, dept_name, salary)$



Example

- $R = (A, B, C)$
 $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (B, C)$
 - Lossless-join decomposition:
$$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
 - Dependency preserving
- $R_1 = (A, B), R_2 = (A, C)$
 - Lossless-join decomposition:
$$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
 - Not dependency preserving
(cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)



BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider a schema:
 $dept_advisor(s_ID, i_ID, department_name)$
- With function dependencies:
 $i_ID \rightarrow dept_name$
 $s_ID, dept_name \rightarrow i_ID$
- $dept_advisor$ is not in BCNF
 - i_ID is not a superkey.
- Any decomposition of $dept_advisor$ will not include all the attributes in
 $s_ID, dept_name \rightarrow i_ID$
- Thus, the composition is NOT be dependency preserving



Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:

$$\alpha \rightarrow \beta \text{ in } F^+$$

at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
- α is a superkey for R
- Each attribute A in $\beta - \alpha$ is contained in a candidate key for R .

(**NOTE:** each attribute may be in a different candidate key)

- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).



3NF Example

- Consider a schema:
 $dept_advisor(s_ID, i_ID, dept_name)$
- With function dependencies:
 $i_ID \rightarrow dept_name$
 $s_ID, dept_name \rightarrow i_ID$
- Two candidate keys = $\{s_ID, dept_name\}, \{s_ID, i_ID\}$
- We have seen before that $dept_advisor$ is **not** in BCNF
- R , however, is in 3NF
 - $s_ID, dept_name$ is a superkey
 - $i_ID \rightarrow dept_name$ and i_ID is NOT a superkey, but:
 - $\{dept_name\} - \{i_ID\} = \{dept_name\}$ and
 - $dept_name$ is contained in a candidate key



Redundancy in 3NF

- Consider the schema R below, which is in 3NF
 - $R = (J, K, L)$
 - $F = \{JK \rightarrow L, L \rightarrow K\}$
 - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
<i>null</i>	l_2	k_2

- What is wrong with the table?
 - Repetition of information
 - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)



Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF. It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Disadvantages to 3NF.
 - We may have to use null values to represent some of the possible meaningful relationships among data items.
 - There is the problem of repetition of information.



Goals of Normalization

- Let R be a relation scheme 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, need to decompose it into a set of relation scheme $\{R_1, R_2, \dots, R_n\}$ such that:
 - Each relation scheme is in good form
 - The decomposition is a lossless decomposition
 - Preferably, the decomposition should be dependency preserving.



How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

inst_info (*ID*, *child_name*, *phone*)

- where an instructor may have more than one phone and can have multiple children
- Instance of *inst_info*

<i>ID</i>	<i>child_name</i>	<i>phone</i>
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321



How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)

(99999, William, 981-992-3443)



Higher Normal Forms

- It is better to decompose *inst_info* into:
 - *inst_child*:

<i>ID</i>	<i>child_name</i>
99999	David
99999	William

- *inst_phone*:

<i>ID</i>	<i>phone</i>
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later



Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i .
 - A decomposition is **dependency preserving**, if

$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$

- Using the above definition, testing for dependency preservation take exponential time.
- Not that if a decomposition is NOT dependency preserving then checking updates for violation of functional dependencies may require computing joins, which is expensive.



Dependency Preservation (Cont.)

- Let F be the set of dependencies on schema R and let R_1, R_2, \dots, R_n be a decomposition of R .
- The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include **only** attributes of R_i .
- Since all functional dependencies in a restriction involve attributes of only one relation schema, it is possible to test such a dependency for satisfaction by checking only one relation.
- Note that the definition of restriction uses all dependencies in F^+ , not just those in F .
- The set of restrictions F_1, F_2, \dots, F_n is the set of functional dependencies that can be checked efficiently.



Example

- $R = (A, B, C)$
 $F = \{A \rightarrow B$
 $B \rightarrow C\}$
Key = $\{A\}$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
 - R_1 and R_2 in BCNF
 - Lossless-join decomposition
 - Dependency preserving



Third Normal Form

- There are some situations where
 - BCNF is not dependency preserving, and
 - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
 - Allows some redundancy (with resultant problems; we will see examples later)
 - But functional dependencies can be checked on individual relations without computing a join.
 - There is always a lossless-join, dependency-preserving decomposition into 3NF.



3NF Example -- Relation *dept_advisor*

- *dept_advisor* (*s_ID*, *i_ID*, *dept_name*)
 $F = \{s_ID, dept_name \rightarrow i_ID, i_ID \rightarrow dept_name\}$
- Two candidate keys: *s_ID, dept_name*, and *i_ID, s_ID*
- *R* is in 3NF
 - $s_ID, dept_name \rightarrow i_ID \quad s_ID$
 - *dept_name* is a superkey
 - $i_ID \rightarrow dept_name$
 - *dept_name* is contained in a candidate key



Testing for 3NF

- Need to check only FDs in F , need not check all FDs in F^+ .
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of R
 - This test is rather more expensive, since it involve finding candidate keys
 - Testing for 3NF has been shown to be NP-hard
 - Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time



Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
 - The decomposition is lossless
 - The dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
 - The decomposition is lossless
 - It may not be possible to preserve dependencies.



Design Goals

- Goal for a relational database design is:
 - BCNF.
 - Lossless join.
 - Dependency preservation.
- If we cannot achieve this, we accept one of
 - Lack of dependency preservation
 - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.



Multivalued Dependencies (MVDs)

- Suppose we record names of children, and phone numbers for instructors:
 - *inst_child*(*ID*, *child_name*)
 - *inst_phone*(*ID*, *phone_number*)
- If we were to combine these schemas to get
 - *inst_info*(*ID*, *child_name*, *phone_number*)
 - Example data:
 - (99999, David, 512-555-1234)
 - (99999, David, 512-555-4321)
 - (99999, William, 512-555-1234)
 - (99999, William, 512-555-4321)
- This relation is in BCNF
 - Why?



Multivalued Dependencies

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \twoheadrightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$t_3[\beta] = t_1[\beta]$$

$$t_3[R - \beta] = t_2[R - \beta]$$

$$t_4[\beta] = t_2[\beta]$$

$$t_4[R - \beta] = t_1[R - \beta]$$



MVD -- Tabular representation

- Tabular representation of $\alpha \twoheadrightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$



MVD (Cont.)

- Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

Y, Z, W

- We say that $Y \twoheadrightarrow Z$ (Y **multidetermines** Z) if and only if for all possible relations $r(R)$

$$\langle y_1, z_1, w_1 \rangle \in r \text{ and } \langle y_1, z_2, w_2 \rangle \in r$$

then

$$\langle y_1, z_1, w_2 \rangle \in r \text{ and } \langle y_1, z_2, w_1 \rangle \in r$$

- Note that since the behavior of Z and W are identical it follows that $Y \twoheadrightarrow Z$ if $Y \twoheadrightarrow W$



Example

- In our example:

$ID \twoheadrightarrow child_name$

$ID \twoheadrightarrow phone_number$

- The above formal definition is supposed to formalize the notion that given a particular value of Y (ID) it has associated with it a set of values of Z ($child_name$) and a set of values of W ($phone_number$), and these two sets are in some sense independent of each other.
- Note:
 - If $Y \rightarrow Z$ then $Y \twoheadrightarrow Z$
 - Indeed we have (in above notation) $Z_1 = Z_2$
The claim follows.



Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
 1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
 2. To specify **constraints** on the set of legal relations. We shall concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation r fails to satisfy a given multivalued dependency, we can construct a relations r' that does satisfy the multivalued dependency by adding tuples to r .



Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:

- If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$

That is, every functional dependency is also a multivalued dependency

- The **closure** D^+ of D is the set of all functional and multivalued dependencies logically implied by D .
 - We can compute D^+ from D , using the formal definitions of functional dependencies and multivalued dependencies.
 - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
 - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules (Appendix C).



Fourth Normal Form

- A relation schema R is in **4NF** with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
 - $\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
 - α is a superkey for schema R
- If a relation is in 4NF it is in BCNF



Restriction of Multivalued Dependencies

- The restriction of D to R_i is the set D_i consisting of
 - All functional dependencies in D^+ that include only attributes of R_i
 - All multivalued dependencies of the form

$$\alpha \twoheadrightarrow (\beta \cap R_i)$$

where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in D^+



Example

- $R = (A, B, C, G, H, I)$
 $F = \{ A \twoheadrightarrow B$
 $B \twoheadrightarrow HI$
 $CG \twoheadrightarrow H \}$
- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R
- Decomposition
 - a) $R_1 = (A, B)$ (R_1 is in 4NF)
 - b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF, decompose into R_3 and R_4)
 - c) $R_3 = (C, G, H)$ (R_3 is in 4NF)
 - d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF, decompose into R_5 and R_6)
 - $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI \Rightarrow A \twoheadrightarrow HI$, (MVD transitivity), and
 - and hence $A \twoheadrightarrow I$ (MVD restriction to R_4)
 - e) $R_5 = (A, I)$ (R_5 is in 4NF)
 - f) $R_6 = (A, C, G)$ (R_6 is in 4NF)



5NF or PJNF

- **Join dependencies** generalize multivalued dependencies
 - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used



5NF or PJNF

- A **join dependency (JD)**, denoted by $JD(R_1, R_2, \dots, R_n)$, specified on relation schema R , specifies a constraint on the states r of R . The constraint states that every legal state r of R should have a non-additive join decomposition into R_1, R_2, \dots, R_n ; that is, for every such r we have

$$* (\pi_{R_1}(r), \pi_{R_2}(r), \dots, \pi_{R_n}(r)) = r$$

Note: an MVD is a special case of a JD where $n = 2$.

- A join dependency $JD(R_1, R_2, \dots, R_n)$, specified on relation schema R , is a **trivial JD** if one of the relation schemas R_i in $JD(R_1, R_2, \dots, R_n)$ is equal to R .



5NF or PJNF

- A relation schema R is in **fifth normal form (5NF)** (or **Project-Join Normal Form (PJNF)**) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency $JD(R_1, R_2, \dots, R_n)$ in F^+ (that is, implied by F), every R_i is a superkey of R .



Domain-Key Normal Form (DKNF)

- **Definition:** A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation.
- The **idea** is to specify (theoretically, at least) the “*ultimate normal form*” that takes into account all possible types of dependencies and constraints. .
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.
- The practical utility of DKNF is limited