Chapter 7: Relational Database Design

- Pitfalls in Relational Database Design
- Decomposition
- Normalization Using Functional Dependencies
- Normalization Using Multivalued Dependencies
- Normalization Using Join Dependencies
- Domain-Key Normal Form
- Alternative Approaches to Database Design

Pitfalls in Relational Database Design

- Relational database design requires that we find a “good” collection of relation schemas. A bad design may lead to
  - Repetition of information.
  - Inability to represent certain information.
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Facilitate the checking of updates for violation of database integrity constraints

Example

- Consider the relation schema:

  Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)

- Redundancy:
  - Data for *branch-name, branch-city, assets* are repeated for each loan that a branch makes
  - Wastes space and complicates updating
- Null values
  - Cannot store information about a branch if no loans exist
  - Can use null values, but they are difficult to handle

Decomposition

- Decompose the relation schema *Lending-schema* into:

Branch-customer-schema = (*branch-name, branch-city, assets, customer-name*)

Customer-loan-schema = (*customer-name, loan-number, amount*)

- All attributes of an original schema (R) must appear in the decomposition (R_1, R_2):

$$R = R_1 \cup R_2$$

- Lossless-join decomposition.
  For all possible relations r on schema R

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

Example of a Non Lossless-Join Decomposition

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

  $R_1 = (A) \qquad R_2 = (B)$

A	B
α	1
α	2
β	1

r

A
α
β

$\Pi_A(r)$

B
1
2

$\Pi_{B}(r)$

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

A	B
α	1
α	2
β	1
β	2

Goal — Devise a Theory for the Following:

- Decide whether a particular relation R is in “good” form.
- In the case that a relation R is not in “good” form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Normalization Using Functional Dependencies

When we decompose a relation schema R with a set of functional dependencies F into R_1 and R_2 we want:

- Lossless-join decomposition: At least one of the following dependencies is in F+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- No redundancy: The relations R_1 and R_2 preferably should be in either Boyce-Codd Normal Form or Third Normal Form.
- Dependency preservation: Let F_i be the set of dependencies in F^+ that include only attributes in R_i. Test to see if:
  - $(F_1 \cup F_2)^+ = F^+$

  Otherwise, checking updates for violation of functional dependencies is expensive.

Example

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

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*

Example

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

  $F = \{A \rightarrow B$

  $\quad B \rightarrow C\}$

  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

BCNF Decomposition Algorithm

result := $\{R\}$;
done := false;
compute F^+;
while (**not** *done*) **do**
  if (there is a schema R_i in *result* that is not in BCNF)
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial functional
       dependency that holds on R_i
       such that $\alpha \rightarrow R_i$ is not in F^+,
       and $\alpha \cap \beta = \emptyset$;
      result := (*result* $-$ R_i) $\cup$($R_i - \beta$) $\cup$(α, β);
    end
    else *done* := true;

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- R = (*branch-name, branch-city, assets,*
  customer-name, loan-number, amount)

  $F = \{$*branch-name* $\rightarrow$ *assets branch-city*
  loan-number $\rightarrow$ *amount branch-name*$\}$

  Key = $\{$*loan-number, customer-name*$\}$

- Decomposition
  - R_1 = (*branch-name, branch-city, assets*)
  - R_2 = (*branch-name, customer-name, loan-number, amount*)
  - R_3 = (*branch-name, loan-number, amount*)
  - R_4 = (*customer-name, loan-number*)

- Final decomposition

$$R_1, R_3, R_4$$

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

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*.

- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).

3NF (Cont.)

- Example
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$
  - Two candidate keys: JK and JL
  - R is in 3NF

$JK \rightarrow L$	JK is a superkey
$L \rightarrow K$	K is contained in a candidate key

- Algorithm to decompose a relation schema R into a set of relation schemas $\{R_1, R_2, ..., R_n\}$ such that:
  - each relation schema R_i is in 3NF
  - lossless-join decomposition
  - dependency preserving

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
  if none of the schemas $R_j,\ 1 \leq j \leq i$ contains $\alpha\,\beta$
    then begin
      $i := i + 1$;
      $R_i := \alpha\,\beta$;
    end
if none of the schemas $R_j,\ 1 \leq j \leq i$ contains
a candidate key for R
  then begin
    $i := i + 1$;
    $R_i :=$ any candidate key for R;
  end
return $(R_1,\ R_2, ..., R_i)$

Example

- Relation schema:

  Banker-info-schema = (*branch-name*, *customer-name*,
  banker-name, *office-number*)

- The functional dependencies for this relation schema are:

  banker-name $\rightarrow$ *branch-name office-number*
  customer-name branch-name $\rightarrow$ *banker-name*

- The key is:

  $\{$*customer-name*, *branch-name*$\}$

Applying 3NF to *Banker – info – schema*

- The **for** loop in the algorithm causes us to include the following schemas in our decomposition:

  Banker-office-schema = (*banker-name*, *branch-name*, *office-number*)
  Banker-schema = (*customer-name*, *branch-name*, *banker-name*)

- Since *Banker-schema* contains a candidate key for *Banker-info-schema*, we are done with the decomposition process.

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - dependencies are preserved
- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies

Comparison of BCNF and 3NF (Cont.)

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
- Consider the following relation

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- A schema that is in 3NF but not in BCNF has the problems of
  - repetition of information (e.g., the relationship l_1, k_1)
  - need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept:
  - 3NF.
  - Lossless join.
  - Dependency preservation.

Normalization Using Multivalued Dependencies

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

  $$\textit{classes}(\textit{course, teacher, book})$$

  such that $(c,t,b) \in \textit{classes}$ means that t is qualified to teach c, and b is a required textbook for c
- The database is supposed to list for each course the set of teachers any one of which can be the course's instructor, and the set of books, all of which are required for the course (no matter who teaches it).

course	*teacher*	*book*
database	Avi	Korth
database	Avi	Ullman
database	Hank	Korth
database	Hank	Ullman
database	Sudarshan	Korth
database	Sudarshan	Ullman
operating systems	Avi	Silberschatz
operating systems	Avi	Shaw
operating systems	Jim	Silberschatz
operating systems	Jim	Shaw

classes

- Since there are no non-trivial dependencies, (*course, teacher, book*) is the only key, and therefore the relation is in BCNF
- Insertion anomalies – i.e., if Sara is a new teacher that can teach database, two tuples need to be inserted

(database, Sara, Korth)
(database, Sara, Ullman)

- Therefore, it is better to decompose *classes* into:

course	*teacher*
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

teaches

course	*book*
database	Korth
database	Ullman
operating systems	Silberschatz
operating systems	Shaw

text

- We shall see that these two relations are in Fourth Normal Form (4NF)

Multivalued Dependencies (MVDs)

- 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 of 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:

$$\begin{aligned}
t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\
t_3[\beta] &= t_1[\beta] \\
t_3[R - \beta] &= t_2[R - \beta] \\
t_4[\beta] &= t_2[\beta] \\
t_4[R - \beta] &= t_1[R - \beta]
\end{aligned}$$

MVD (Cont.)

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

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

Example

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

$$Y, Z, W$$

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

$$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$

  then

$$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

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

Example (Cont.)

- In our example:

$$\textit{course} \rightarrow\rightarrow \textit{teacher}$$
$$\textit{course} \rightarrow\rightarrow \textit{book}$$

- The above formal definition is supposed to formalize the notion that given a particular value of Y (*course*) it has associated with it a set of values of Z (*teacher*) and a set of values of W (*book*), and these two sets are in some sense independent of each other.

- Note:
  - If $Y \rightarrow Z$ then $Y \rightarrow\rightarrow Z$
  - 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 thus 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 relation r' that does satisfy the multivalued dependency by adding tuples to r.

Theory of Multivalued Dependencies

- Let *D* denote a set of functional and multivalued dependencies. The closure D^+ of *D* is the set of all functional and multivalued dependencies logically implied by *D*.
- Sound and complete inference rules for functional and multivalued dependencies:
  1. **Reflexivity rule**. If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
  2. **Augmentation rule**. If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.
  3. **Transitivity rule**. If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

Theory of Multivalued Dependencies (Cont.)

4. **Complementation rule**. If $\alpha \twoheadrightarrow \beta$ holds, then $\alpha \twoheadrightarrow R - \beta - \alpha$ holds.

5. **Multivalued augmentation rule**. If $\alpha \twoheadrightarrow \beta$ holds and $\gamma \subseteq R$ and $\delta \subseteq \gamma$, then $\gamma\alpha \twoheadrightarrow \delta\beta$ holds.

6. **Multivalued transitivity rule**. If $\alpha \twoheadrightarrow \beta$ holds and $\beta \twoheadrightarrow \gamma$ holds, then $\alpha \twoheadrightarrow \gamma - \beta$ holds.

7. **Replication rule**. If $\alpha \rightarrow \beta$ holds, then $\alpha \twoheadrightarrow \beta$.

8. **Coalescence rule**. If $\alpha \twoheadrightarrow \beta$ holds and $\gamma \subseteq \beta$ and there is a δ such that $\delta \subseteq R$ and $\delta \cap \beta = \emptyset$ and $\delta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ holds.

Simplification of the Computation of D^+

- We can simplify the computation of the closure of *D* by using the following rules (proved using rules 1–8).
  - **Multivalued union rule**. If $\alpha \rightarrow\rightarrow \beta$ holds and $\alpha \rightarrow\rightarrow \gamma$ holds, then $\alpha \rightarrow\rightarrow \beta\gamma$ holds.
  - **Intersection rule**. If $\alpha \rightarrow\rightarrow \beta$ holds and $\alpha \rightarrow\rightarrow \gamma$ holds, then $\alpha \rightarrow\rightarrow \beta \cap \gamma$ holds.
  - **Difference rule**. If $\alpha \rightarrow\rightarrow \beta$ holds and $\alpha \rightarrow\rightarrow \gamma$ holds, then $\alpha \rightarrow\rightarrow \beta - \gamma$ holds and $\alpha \rightarrow\rightarrow \gamma - \beta$ holds.

Example

- $R = (A,\ B,\ C,\ G,\ H,\ I)$

  $D = \{A \rightarrow\!\!\!\rightarrow B$
  $\quad B \rightarrow\!\!\!\rightarrow HI$
  $\quad CG \rightarrow H\}$

- Some members of D^+:
  - $A \rightarrow\!\!\!\rightarrow CGHI$.

    Since $A \rightarrow\!\!\!\rightarrow B$, the complementation rule (4) implies that $A \rightarrow\!\!\!\rightarrow R - B - A$.

    Since $R - B - A = CGHI$, so $A \rightarrow\!\!\!\rightarrow CGHI$.
  - $A \rightarrow\!\!\!\rightarrow HI$.

    Since $A \rightarrow\!\!\!\rightarrow B$ and $B \rightarrow\!\!\!\rightarrow HI$, the multivalued transitivity rule (6) implies that $A \rightarrow\!\!\!\rightarrow HI - B$.

    Since $HI - B = HI$, $A \rightarrow\!\!\!\rightarrow HI$.

Example (Cont.)

- Some members of D^+ (cont.):
  - $B \rightarrow H$.
    Apply the coalescence rule (8); $B \rightarrow\!\!\!\rightarrow HI$ holds.
    Since $H \subseteq HI$ and $CG \rightarrow H$ and $CG \cap HI = \emptyset$, the coalescence rule is satisfied with α being B, β being HI, δ being CG, and γ being H. We conclude that $B \rightarrow H$.
  - $A \rightarrow\!\!\!\rightarrow CG$.
    $A \rightarrow\!\!\!\rightarrow CGHI$ and $A \rightarrow\!\!\!\rightarrow HI$.
    By the difference rule, $A \rightarrow\!\!\!\rightarrow CGHI - HI$.
    Since $CGHI - HI = CG$, $A \rightarrow\!\!\!\rightarrow CG$.

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

4NF Decomposition Algorithm

result := $\{R\}$;
done := false;
compute F^+;
while (**not** *done*) **do**
if (there is a schema R_i in *result* that is not in 4NF)
then begin
let $\alpha \rightarrow \beta$ be a nontrivial multivalued dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$;
result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
end
else *done* := true;

Note: each R_i is in 4NF, and decomposition is lossless-join.

Example

- $R = (A,\ B,\ C,\ G,\ H,\ I)$

  $$F = \{A \rightarrow\!\!\!\rightarrow B \\ B \rightarrow\!\!\!\rightarrow HI \\ CG \rightarrow H\}$$

- R is not in 4NF since $A \rightarrow\!\!\!\rightarrow B$ 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)
  c) $R_3 = (C,\ G,\ H)$ (R_3 is in 4NF)
  d) $R_4 = (A,\ C,\ G,\ I)$ (R_4 is not in 4NF)

- Since $A \rightarrow\!\!\!\rightarrow B$ and $B \rightarrow\!\!\!\rightarrow HI$, $A \rightarrow\!\!\!\rightarrow HI$, $A \rightarrow\!\!\!\rightarrow I$

  e) $R_5 = (A,\ I)$ (R_5 is in 4NF)
  f) $R_6 = (A,\ C,\ G)$ (R_6 is in 4NF)

Multivalued Dependency Preservation

- Let $R_1, R_2, \ldots, R_n$ be a decomposition of R, and D a set of both functional and 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^+

- The decomposition is *dependency-preserving* with respect to D if, for every set of relations $r_1(R_1), r_2(R_2), \ldots, r_n(R_n)$ such that for all i, r_i satisfies D_i, there exists a relation $r(R)$ that satisfies D and for which $r_i = \Pi_{R_i}(r)$ for all i.

- Decomposition into 4NF may not be dependency preserving (even on just the multivalued dependencies)

Normalization Using Join Dependencies

- Join dependencies constrain the set of legal relations over a schema R to those relations for which a given decomposition is a lossless-join decomposition.

- Let R be a relation schema and $R_1, R_2, ..., R_n$ be a decomposition of R. If $R = R_1 \cup R_2 \cup ... \cup R_n$, we say that a relation $r(R)$ satisfies the *join dependency* $*(R_1, R_2, ..., R_n)$ if:

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

  A join dependency is *trivial* if one of the R_i is R itself.

- A join dependency $*(R_1, R_2)$ is equivalent to the multivalued dependency $R_1 \cap R_2 \rightarrow\rightarrow R_2$. Conversely, $\alpha \rightarrow\rightarrow \beta$ is equivalent to $*(\alpha \cup (R - \beta), \alpha \cup \beta)$

- However, there are join dependencies that are not equivalent to any multivalued dependency.

Project-Join Normal Form (PJNF)

- A relation schema R is in PJNF with respect to a set D of functional, multivalued, and join dependencies if for all join dependencies in $D+$ of the form

$$*(R_1, R_2, ..., R_n) \text{ where each } R_i \subseteq R$$
$$\text{and } R = R_1 \cup R_2 \cup ... \cup R_n,$$

  at least one of the following holds:

  - $*(R_1, R_2, ..., R_n)$ is a trivial join dependency.
  - Every R_i is a superkey for R.

- Since every multivalued dependency is also a join dependency, every PJNF schema is also in 4NF.

Example

- Consider *Loan-info-schema* = (*branch-name, customer-name, loan-number, amount*).
- Each loan has one or more customers, is in one or more branches and has a loan amount; these relationships are independent, hence we have the join dependency

  *((*loan-number, branch-name*), (*loan-number, customer-name*), (*loan-number, amount*))
- *Loan-info-schema* is not in PJNF with respect to the set of dependencies containing the above join dependency. To put *Loan-info-schema* into PJNF, we must decompose it into the three schemas specified by the join dependency:
  - (*loan-number, branch-name*)
  - (*loan-number, customer-name*)
  - (*loan-number, amount*)

Domain-Key Normal Form (DKNY)

- **Domain declaration**. Let A be an attribute, and let **dom** be a set of values. The domain declaration $A \subseteq$ **dom** requires that the A value of all tuples be values in **dom**.
- **Key declaration**. Let R be a relation schema with $K \subseteq R$. The key declaration **key** (K) requires that K be a superkey for schema R ($K \rightarrow R$). All key declarations are functional dependencies but not all functional dependencies are key declarations.
- **General constraint**. A general constraint is a predicate on the set of all relations on a given schema.
- Let **D** be a set of domain constraints and let **K** be a set of key constraints for a relation schema R. Let **G** denote the general constraints for R. Schema R is in DKNF if **D** $\cup$ **K** logically imply **G**.

Example

- Accounts whose *account-number* begins with the digit 9 are special high-interest accounts with a minimum balance of $2500.
- General constraint: "If the first digit of t[*account-number*] is 9, then t[*balance*] $\geq$ 2500."
- DKNF design:

  Regular-acct-schema = (*branch-name, account-number, balance*)
  Special-acct-schema = (*branch-name, account-number, balance*)

- Domain constraints for *Special-acct-schema* require that for each account:
  - The account number begins with 9.
  - The balance is greater than 2500.

DKNF rephrasing of PJNF Definition

- Let $R = (A_1, A_2, ..., A_n)$ be a relation schema. Let dom(A_i) denote the domain of attribute A_i, and let all these domains be infinite. Then all domain constraints **D** are of the form $A_i \subseteq \text{dom}(A_i)$.
- Let the general constraints be a set **G** of functional, multivalued, or join dependencies. If F is the set of functional dependencies in **G**, let the set **K** of key constraints be those nontrivial functional dependencies in F^+ of the form $\alpha \rightarrow R$.
- Schema R is in PJNF if and only if it is in DKNF with respect to **D**, **K**, and **G**.

Alternative Approaches to Database Design

- *Dangling tuples* —Tuples that "disappear" in computing a join.
  - Let $r_1(R_1),\ r_2(R_2), ..., r_n(R_n)$ be a set of relations.
  - A tuple t of relation r_i is a *dangling tuple* if t is not in the relation:

$$\Pi_{R_i}\ (r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n)$$

- The relation $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$ is called a *universal relation* since it involves all the attributes in the "universe" defined by $R_1 \cup R_2 \cup \ldots \cup R_n$.
- If dangling tuples are allowed in the database, instead of decomposing a universal relation, we may prefer to *synthesize* a collection of normal form schemas from a given set of attributes.