Database Systems (CSF212)
Lecture – 22

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

Relational Database Design

How good is BCNF?

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

    classes (*course, teacher, book*)

  such that (*c, t, b*) $\in$ *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).

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a database ***classes*** (*course, teacher, book*)

  such that (*c, t, b*) $\in$ *classes* means that *t* is qualified to teach only the course *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)

How good is BCNF? (Cont.)

course	*teacher*	*book*
database	Tom	DB Concepts
database	John	Ullman
database	Tom	Ullman
database	John	DBConcepts
operating systems	Pete	OS Concepts
operating systems	Pete	Stallings

classes

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if Marilyn is a new teacher that can teach database, two tuples need to be inserted (database, Marilyn, DB Concepts)
  (database, Marilyn, Ullman)

How good is BCNF? (Cont.)

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

course	***teacher***
database	John
database	Tom
operating systems	Pete

teaches

course	***book***
database	DB Concepts
database	Ullman
operating systems	OS Concepts
operating systems	Stallings

text

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

Multivalued Dependencies (MVDs)

- A **multivalued dependency** (**MVD**) $X \longrightarrow\!\!> Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R - (X + Y))$:
  - $t_3[X] = t_4[X] = t_1[X] = t_2[X]$
  - $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$
  - $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

- An MVD $X \longrightarrow\!\!> Y$ in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) $X \cup Y = R$.

MVD (Cont.)

	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	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Tabular representation of $\alpha \rightarrow\rightarrow \beta$

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 multi determines Z) if and only if for all possible relations $r(R)$

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

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

- Note that since the behavior of Z and W are identical it follows that

$$Y \rightarrow\rightarrow Z \text{ if } Y \rightarrow\rightarrow W$$

basically 4 tuples honge, sab mein ek value constant. 2 mein second constant par third alag. 2 mein 2nd alag par 3rd constant.

Example

- In our example:

  course $\rightarrow\rightarrow$ teacher
  course $\rightarrow\rightarrow$ *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

Example

- In our example:

  $course \rightarrow\rightarrow$ teacher
  $course \rightarrow\rightarrow book$

- 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

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 relations *r*´ that does satisfy the multivalued dependency by adding tuples to *r*.

Theory of MVDs

- The **closure** D^+ of D is the set of all functional and multivalued dependencies logically implied by D and can be computed using the IR rules defined for functional dependencies and multivalued dependencies:
  - **IR4** (complementation rule for MVDs): $\{X \rightarrow\rightarrow Y\} \models \{X \rightarrow\rightarrow (R - (X \cup Y))\}$
  - **IR5** (augmentation rule for MVDs): If $\{X \rightarrow\rightarrow Y\}$ and $Z \subseteq W$, then $WX \rightarrow\rightarrow YZ$
  - **IR6** (transitive rule for MVDs): $\{X \rightarrow\rightarrow Y, Y \rightarrow\rightarrow Z\} \models X \rightarrow\rightarrow (Z - Y)$
  - **IR7** (replication rule for FD to MVD): $\{X \rightarrow Y\} \models X \rightarrow\rightarrow Y$
  - ***IR8*** (coalescence rule for FDs and MVDs): If $X \rightarrow\rightarrow Y$ and there exists W with the properties that (a) $W \cap Y = \emptyset$ (b) $W \rightarrow Z$, and (c) $Z \subseteq Y$, then $X \rightarrow Z$.

Theory of MVDs

– We can simplify calculating , the closure of *D* by using the following rules, derivable from the previous ones

- **Multi valued union rule**
  - $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z\} \models X \twoheadrightarrow YZ$
- **Multi valued intersection rule**
  - $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z\} \models X \twoheadrightarrow Y \cap Z$
- **Multi valued Difference rule**
  - $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z\} \models X \twoheadrightarrow Y - Z$ and $X \twoheadrightarrow Z - Y$

Example

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

  $F = \{ A \rightarrow\rightarrow B, \quad B \rightarrow\rightarrow HI, \quad CG \rightarrow\rightarrow H \}$
- *Some examples of D+ are:*
- $A \rightarrow\rightarrow CGHI$ (since $A \rightarrow\rightarrow B$, complementation rule implies that , $A \rightarrow\rightarrow$ R – B - A

- $A \rightarrow\rightarrow HI$ *(multi valued transitivity* $A \rightarrow\rightarrow B$, $B \rightarrow\rightarrow HI$*)*

- $B \rightarrow\rightarrow H$ (coalescence rule can be applied. $B \rightarrow\rightarrow HI$ holds, H ⊆ HI and $CG \rightarrow\rightarrow H$ and CG ∩ HI = ঌ}

- $A \rightarrow\rightarrow CG$ *(*$A \rightarrow\rightarrow CGHI$ *and* $A \rightarrow\rightarrow HI$*, by difference rule,* $A \rightarrow\rightarrow CGHI - HI$*)*

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
- 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

    $\alpha \rightarrow\rightarrow (\beta \cap R_i)$, where $\alpha \subseteq R_i$ and $\alpha \rightarrow\rightarrow \beta$ is in D^+

4NF Decomposition Algorithm

result: = {*R*};
done := false;
compute D^+;
Let D_i denote the restriction of D^+ to R_i
while (**not** *done*) **{**
if (there is a schema $\mathbf{R}_i$ in *result* that is not in 4NF)
then
begin
let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \phi$;
result := (*result* - R_i) $\cup$ (R_i - β) $\cup$ (α, β);
end
else
done:= true;
}

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow\rightarrow B, \quad B \rightarrow\rightarrow HI, \quad CG \rightarrow\rightarrow H \}$
- 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)

Join Dependencies and Fifth Normal Form

<u>Definition:</u>

- A **join dependency (JD)**, denoted by JD(R_1, R_2, ..., 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, ..., R_n; that is, for every such r we have
  - $$*(\pi_{R1}(r), \pi_{R2}(r), \ldots, \pi_{Rn}(r)) = r$$

  ***Note**: an MVD is a special case of a JD where n = 2.*
- A join dependency JD(R_1, R_2, ..., R_n), specified on relation schema R, is a **trivial JD** if one of the relation schemas

Join Dependencies and Fifth Normal Form

5NF Definition:

- 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, ..., R_n) in F^+ every R_i is a superkey of R.

- Example: **R =** (A, B, C, D, E, I)

  F = {

  *(A B C D, C D E, B D I),

  *(A B, B C D, A D)

  $A \rightarrow B\ C\ D\ E$,

  $B\ C \rightarrow A\ I$

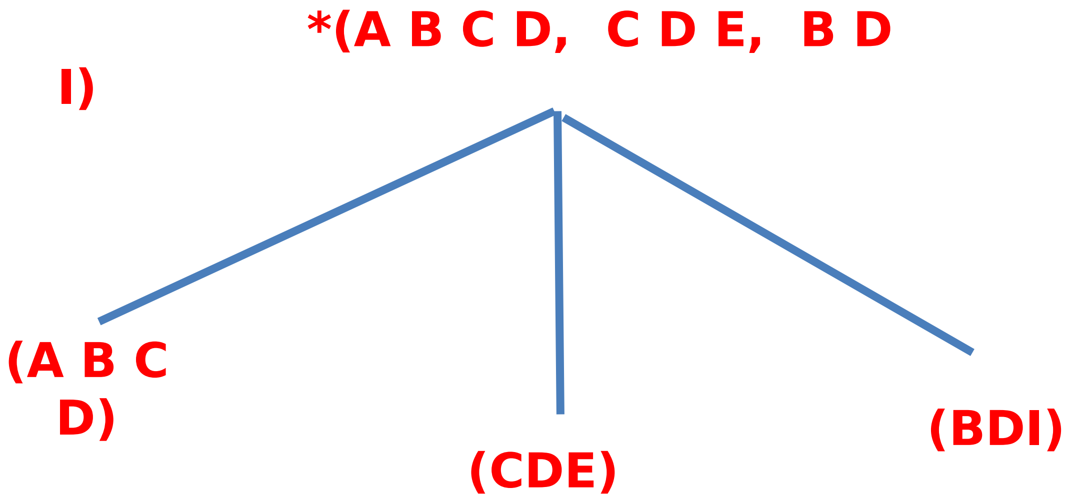
  }

Join Dependencies and Fifth Normal Form

F = { *(A B C D, C D E, B D I), *(A B, B C D, A D),
$A \rightarrow B\ C\ D\ E$, $B\ C \rightarrow A\ I$ }

keys = {A, BC}

None of the FD's are violating the concitions of BCNF.



*(A B, B C D, A D) applies only on the schema (ABCD) and all R_i's are the super key of the schema (ABCD)

Join Dependencies and Fifth Normal Form

- Discovering join dependencies in practical databases with hundreds of relations is next to impossible. Therefore, 5NF is rarely used in practice

- **Domain-key normal form (DKNF):** 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

- It might not be possible to specify every constraint through domain and key constraints only. For example sometimes it is difficult to even specify general integrity constraints in terms of domain and key constraints. So, the practical utility of DKNF is

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables.
  - *R* could have been a single relation containing *all* attributes that are of interest (called **universal relation**).
  - Normalization breaks *R* into smaller relations.
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes *department_number* and *department_address*, and a functional dependency

    $$department\_number \rightarrow department\_address$$

  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but are rare

De-normalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying *customer_name* along with *account_number* and *balance* requires join of *account* with *depositor*
- Alternative 1: Use denormalized relation containing attributes of *account* as well as *depositor* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as
  account-depositor
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors