Database Management Systems

Database Normalization

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
and
Centre for Artificial Intelligence and Machine Learning
Indian Statistical Institute, Kolkata

March, 2022

1 Data Redundancy

2 Normalization and Denormalization

3 Normal Forms
- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form
- Elementary Key Normal Form
- Fourth Normal Form
- Fifth Normal Form
- Domain Key Normal Form
- Sixth Normal Form

Redundancy in databases

Redundancy in a database denotes the repetition of stored data

Redundancy might cause various anomalies and problems pertaining to storage requirements:

- Insertion anomalies: It may be impossible to store certain information without storing some other, unrelated information.
- Deletion anomalies: It may be impossible to delete certain information without losing some other, unrelated information.
- Update anomalies: If one copy of such repeated data is updated, all copies need to be updated to prevent inconsistency.
- Increasing storage requirements: The storage requirements may increase over time.

Redundancy in databases

Redundancy in a database denotes the repetition of stored data

Redundancy might cause various anomalies and problems pertaining to storage requirements:

- Insertion anomalies: It may be impossible to store certain information without storing some other, unrelated information.
- Deletion anomalies: It may be impossible to delete certain information without losing some other, unrelated information.
- Update anomalies: If one copy of such repeated data is updated, all copies need to be updated to prevent inconsistency.
- Increasing storage requirements: The storage requirements may increase over time.

These issues can be addressed by decomposing the database – normalization forces this!!!

Insertion anomaly – An example

Consider the following table (the attributes are `not null`) detailing some of the cars available in the Kolkata market.

Company	Country	Make	Distributor
Maruti	India	WagonR	Carwala
Maruti	India	WagonR	Bhalla
Toyota	Japan	RAV4	CarTrade
BMW	Germany	X1	CarTrade

Suppose Tesla, a company from US, is now collaborating with Toyota to bring the make RAV4 in the Kolkata market with no distributor announced yet.

This insertion is not possible in the above table as the `Distributor` cannot be null.

Deletion anomaly – An example

Consider the following table (the attributes are `not null`) detailing some of the cars available in the Kolkata market.

Company	**Country**	**Make**	**Distributor**
Maruti	India	WagonR	Carwala
Maruti	India	WagonR	Bhalla
Toyota	Japan	RAV4	CarTrade
BMW	Germany	X1	CarTrade

Suppose CarTrade is no more a distributor for the make X1 of BMW, a company from Germany.

This deletion from the above table would result in the car record being deleted.

Update anomaly – An example

Consider the following table (the attributes are `not null`) detailing some of the cars available in the Kolkata market.

Company	Country	Make	Distributor
Maruti	India	WagonR	Carwala
Maruti	India	WagonR	Bhalla
Toyota	Japan	RAV4	CarTrade
BMW	Germany	X1	CarTrade

Suppose Maruti is no more an Indian company due to its 100% procurement by Suzuki Motor Corporation, a company from Japan.

This update is to be made in multiple records in the above table resulting into atomicity challenges.

An overview of different normal forms in the literature

Normal Form	Details	Reference
1NF (Codd (1970), Date (2006))	Domains should be atomic/At least one candidate key	[1, 9]
2NF (Codd (1971))	No non-prime attribute is functionally dependent on a proper subset of any candidate key	[2]
3NF (Codd (1971), Zaniolo (1982))	Every non-prime attribute is non-transitively dependent on every candidate key	[2, 7]
BCNF (Codd (1974))	Every non-trivial functional dependency is a dependency on a superkey	[3]
EKNF (Zaniolo (1982))	Every non-trivial functional dependency is either the dependency of an elementary key attribute or a dependency on a superkey	[7]
4NF (Fagin (1977))	Every non-trivial multi-valued dependency is a dependency on a superkey	[4]
5NF (Fagin (1979))	Every non-trivial join dependency is implied by the superkeys	[5]
DKNF (Fagin (1981))	Every constraint on the table is a logical consequence of the domain and key constraints	[6]
6NF (Date *et al.* (2002))	No non-trivial join dependencies at all (w.r.t generalized join)	[8]

Motivations behind normalization

Normal Form	Basic Motivation
1NF	Removing non-atomicity
2NF	Removing partial dependency (Part of key attribute ! Non-key attribute)
3NF	Removing transitive dependency (Non-key attribute ! Non-key attribute)
BCNF	Removing any kind of redundancy

Problems with normalization

Denormalization

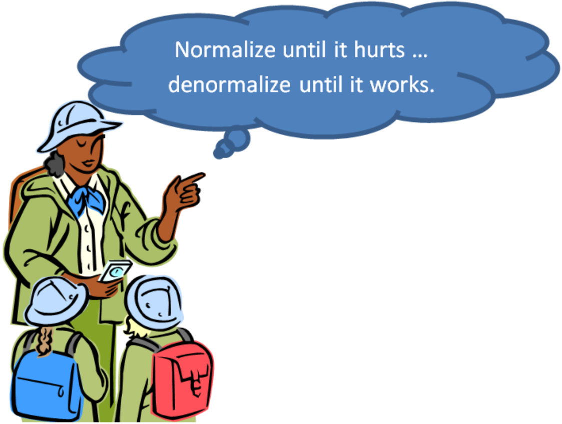
Denormalization is the process of converting a normalized schema to a non-normalized one

Denormalization

Denormalization is the process of converting a normalized schema to a non-normalized one

Note: Designers use denormalization to tune performance of systems to support time-critical operations. They assess the cost, benefit, and risk to identify the right normalization level with respect to the data, its use and its quality requirements.

Normalization versus denormalization



Applications

Normalization:

1. Use of normalization to minimize the impact of various anomalies created with database modification.
2. Use of normalization to reduce the data integrity problems.

Denormalization:

1. Use of denormalization in case the data is not going to be updated after being created.
2. Use of denormalization results into the performance gain.

Note: There is no “ideal” normal form for a table or the data as a whole.

First normal form

The domain (or value set) of an attribute defines the set of values it might contain.

A domain is *atomic* if elements of the domain are considered to be indivisible units.

Company	Make
Maruti	WagonR, Ertiga
Honda	City
Tesla	RAV4
Toyota	RAV4
BMW	X1

Only `Company` has atomic domain

Company	Make
Maruti	WagonR, Ertiga
Honda	City
Tesla, Toyota	RAV4
BMW	X1

None of the attributes have atomic domains

First normal form

Definition (First normal form (1NF))

A relational schema R is in 1NF iff the domains of all attributes in R are *atomic*.

The advantages of 1NF are as follows:

- It eliminates redundancy
- It eliminates repeating groups.

Note: In practice, 1NF includes a few more practical constraints like each attribute must be unique, no tuples are duplicated, and no columns are duplicated.

First normal form

The following relation is not in 1NF because the attribute `Model` is not atomic.

Company	Country	Make	Model	Distributor
Maruti	India	WagonR	LXI, VXI	Carwala
Maruti	India	Ertiga	VXI	Carwala
Maruti	India	WagonR	LXI	Bhalla
Honda	Japan	City	SV	Bhalla
Tesla	USA	RAV4	EV	CarTrade
Toyota	Japan	RAV4	EV	CarTrade
BMW	Germany	X1	Expedition	CarTrade

We can convert this relation into 1NF in two ways!!!

First normal form

Approach 1: Break the tuples containing non-atomic values into multiple tuples.

Company	Country	Make	Model	Distributor
Maruti	India	WagonR	LXI	Carwala
Maruti	India	WagonR	VXI	Carwala
Maruti	India	Ertiga	VXI	Carwala
Maruti	India	WagonR	LXI	Bhalla
Honda	Japan	City	SV	Bhalla
Tesla	USA	RAV4	EV	CarTrade
Toyota	Japan	RAV4	EV	CarTrade
BMW	Germany	X1	Expedition	CarTrade

First normal form

Approach 2: Decompose the relation into multiple relations.

Company	Country	Make
Maruti	India	WagonR
Maruti	India	Ertiga
Honda	Japan	City
Tesla	USA	RAV4
Toyota	Japan	RAV4
BMW	Germany	X1

Make	Model	Distributor
WagonR	LXI	Carwala
WagonR	VXI	Carwala
Ertiga	VXI	Carwala
WagonR	LXI	Bhalla
City	SV	Bhalla
RAV4	EV	CarTrade
X1	Expedition	CarTrade

Why data dependencies are so important?

Choose the best keyset for the locks given below.

Locks	**Keyset 1**	**Keyset 2**	**Keyset 3**
L1	{ K1	{ K1	{ K3
L2	{ K1	{ K2	{ K4
L3	{ K1	{ K3	{ K5
L3	{ K1	{ K4	{ K5

Why data dependencies are so important?

Choose the best keyset for the locks given below.

Locks	Keyset 1	Keyset 2	Keyset 3
L1	{ K1	{ K1	{ K3
L2	{ K1	{ K2	{ K4
L3	{ K1	{ K3	{ K5
L3	{ K1	{ K4	{ K5

- Keyset 1 is not appropriate because a single key can open multiple locks.
- Keyset 2 is not appropriate because the same lock can be opened with multiple keys.
- Keyset 3 is the best option!!!

Functional dependency

Consider a relation schema R. A subset $X \subseteq A(R)$, the attributes in R, is a *superkey* of R if $t1 \neq t2$, for all pairs of tuples $t1, t2 \in R$, implies $t1[X] \neq t2[X]$. This means no two tuples in R may have the same value on attribute set X.

Functional dependency

Consider a relation schema R. A subset $X \subseteq A(R)$, the attributes in R, is a *superkey* of R if $t1 \neq t2$, for all pairs of tuples $t1, t2 \in R$, implies $t1[X] \neq t2[X]$. This means no two tuples in R may have the same value on attribute set X.

The notion of functional dependency generalizes the notion of superkey. Let $X \subseteq A(R)$ and $Y \subseteq A(R)$. The functional dependency $X \rightarrow Y$ holds on schema R if

$$t1[X] = t2[X],$$

in any legal relation $r(R)$, for all pairs of tuples $t1$ and $t2$ in r, then

$$t1[Y] = t2[Y].$$

Superkey versus functional dependency

A	B	C
1	1	2
1	2	2
2	3	1
3	3	1

AB is a superkey
AB ! C holds

A	B	C
1	1	2
1	1	2
2	3	1
3	3	1

AB is not a superkey
AB ! C holds

NOT POSSIBLE

AB is a superkey
AB ! C does not hold

A	B	C
1	1	2
1	1	1
2	3	1
3	3	1

AB is not a superkey
AB ! C does not hold

Partial dependency

The partial dependency $X \to Y$ holds in schema R if there is a $Z \subset X$ such that $Z \to Y$.

We say Y is partially dependent on X if and only if there is a proper subset of X that satisfies the dependency.

Partial dependency

The partial dependency $X \rightarrow Y$ holds in schema R if there is a $Z \subset X$ such that $Z \rightarrow Y$.

We say Y is partially dependent on X if and only if there is a proper subset of X that satisfies the dependency.

X	Y	Z
10	10	20
10	20	20
20	30	10
30	30	10

XY $\rightarrow$ Z is a partial dependency

Note: The dependency $A \rightarrow B$ implies if the A values are same, then the B values are also same.

Second normal form

Definition (Second normal form (2NF))

A relational schema R is in 2NF if each attribute A in R satisfies one of the following criteria:

1. A is part of a candidate key.
2. A is not partially dependent on a candidate key.

In other words, no non-prime attribute (not a part of any candidate key) is dependent on a proper subset of any candidate key.

Note: A *candidate key* is a *superkey* for which no proper subset is a superkey, i.e. a minimal *superkey*.

Second normal form

The following relation is in 1NF but not in 2NF because `Country` is a non-prime attribute that partially depends on `Company`, which is a proper subset of the candidate key *f*`Company`, `Make`, `Model`, `Distributor`*g*.

Company	Country	Make	Model	Distributor
Maruti	India	WagonR	LXI	Carwala
Maruti	India	WagonR	VXI	Carwala
Maruti	India	Ertiga	VXI	Carwala
Maruti	India	WagonR	LXI	Bhalla
Honda	Japan	City	SV	Bhalla
Tesla	USA	RAV4	EV	CarTrade
Toyota	Japan	RAV4	EV	CarTrade
BMW	Germany	X1	Expedition	CarTrade

We can convert this relation into 2NF!!!

Second normal form

Company	Country	Make	Model	Distributor
Maruti	India	WagonR	LXI	Carwala
Maruti	India	WagonR	VXI	Carwala
Maruti	India	Ertiga	VXI	Carwala
Maruti	India	WagonR	LXI	Bhalla
Honda	Japan	City	SV	Bhalla
Tesla	USA	RAV4	EV	CarTrade
Toyota	Japan	RAV4	EV	CarTrade
BMW	Germany	X1	Expedition	CarTrade

- fCompany, Make, Model, Distributorg ! Country
- Company ! Country (Violating 2NF)

Note: Country is partially dependent on fCompany, Make, Model, Distributorg.

Second normal form

Approach: Decompose the relation into multiple relations.

Company	Country
Maruti	India
Honda	Japan
Tesla	USA
Toyota	Japan
BMW	Germany

Company	Make	Model	Distributor
Maruti	WagonR	LXI	Carwala
Maruti	WagonR	VXI	Carwala
Maruti	Ertiga	VXI	Carwala
Maruti	WagonR	LXI	Bhalla
Honda	City	SV	Bhalla
Tesla	RAV4	EV	CarTrade
Toyota	RAV4	EV	CarTrade
BMW	X1	Expedition	CarTrade

Note: Each attribute in the left relation is either a part of the candidate key {`Company`} or having full functional dependency on it, and in the right relation is a part of the candidate key {`Company`, `Make`, `Model`, `Distributor`}.

Functional dependency

Armstrong's axioms:

- **Reflexivity property**: If X is a set of attributes and $Y \subseteq X$, then $X \rightarrow Y$ holds. (known as trivial functional dependency)
- **Augmentation property**: If $X \rightarrow Y$ holds and γ is a set of attributes, then $\gamma X \rightarrow \gamma Y$ holds.
- **Transitivity property**: If both $X \rightarrow Y$ and $Y \rightarrow Z$ holds, then $X \rightarrow Z$ holds.

Functional dependency

Armstrong's axioms:

- **Reflexivity property**: If X is a set of attributes and $Y \subseteq X$, then $X \rightarrow Y$ holds. (known as trivial functional dependency)
- **Augmentation property**: If $X \rightarrow Y$ holds and γ is a set of attributes, then $\gamma X \rightarrow \gamma Y$ holds.
- **Transitivity property**: If both $X \rightarrow Y$ and $Y \rightarrow Z$ holds, then $X \rightarrow Z$ holds.

Other properties:

- **Union property**: If $X \rightarrow Y$ holds and $X \rightarrow Z$ holds, then $X \rightarrow YZ$ holds.
- **Decomposition property**: If $X \rightarrow YZ$ holds, then both $X \rightarrow Y$ and $X \rightarrow Z$ holds.
- **Pseudotransitivity property**: If $X \rightarrow Y$ and $\gamma Y \rightarrow Z$ holds, then $X\gamma \rightarrow Z$ holds.

Closure of functional dependencies (FDs)

We can find F^+, the closure of a set of FDs F, as follows:

Initialize F^+ with F
repeat
  for each functional dependency $f = X \rightarrow Y \in F^+$ **do**
    Apply reflexivity and augmentation properties on f and include the resulting functional dependencies in F^+
  end for
  for each pair of functional dependencies $f_1, f_2 \in F^+$ **do**
    if f_1 and f_2 can be combined together using the transitivity property **then**
      Include the resulting functional dependency in F^+
    end if
  end for
until F^+ does not further change

Closure of functional dependencies (FDs) – An example

Consider a relation *R* = <UVWXYZ> and the set of FDs = *f*U ! V, U ! W, WX ! Y, WX ! Z, V ! Y*g*. Let us compute some non-trivial FDs that can be obtained from this.

- By applying the augmentation property, we obtain
  1. UX ! WX (from U ! W)
  2. WX ! WXZ (from WX ! Z)
  3. WXZ ! YZ (from WX ! Y)
- By applying the transitivity property, we obtain
  1. U ! Y (from U ! V and V ! Y)
  2. UX ! Z (from UX ! WX and WX ! Z)
  3. WX ! YZ (from WX ! WXZ and WXZ ! YZ)

Closure of attribute sets

We can find A^+, the closure of a set of attributes A, as follows:

Initialize A^+ with A
repeat
  for each functional dependency $f = X \rightarrow Y \in F^+$ **do**
    if $X \subseteq A^+$ **then**
      $A^+ \leftarrow A^+ \cup Y$
    end if
  end for
until A^+ does not further change

Note: The closure is defined as the set of attributes that are functionally determined by A under a set of FDs F.

Closure of attribute sets

The usefulness of finding attribute closure is as follows:

- Testing for superkey
  – Compute A^+ and check whether $A(R)$ A^+ and all the tuples are distinct.
- Testing functional dependencies
  – To check if an FD X ! Y holds, just check if Y X^+
  – Same for checking if X ! Y is in F^+ for a given F
- Computing closure of F
  – For each A $A(R)$, we find the closure A^+, and for each S A^+, we output a functional dependency A ! S

Closure of attribute sets

The usefulness of finding attribute closure is as follows:

- Testing for superkey
  – Compute A^{+} and check whether $A(R)$ A^{+} and all the tuples are distinct.
- Testing functional dependencies
  – To check if an FD X ! Y holds, just check if Y X^{+}
  – Same for checking if X ! Y is in F^{+} for a given F
- Computing closure of F
  – For each A $A(R)$, we find the closure A^{+}, and for each S A^{+}, we output a functional dependency A ! S

Note: Even though a subset of attributes X functionally defines the other attributes, it cannot be a superkey until and unless all the tuples are distinct.

Closure of attribute sets – An example

Consider a relation R = <UVWXYZ> and the set of FDs = *f*U ! V, U ! W, WX ! Y, WX ! Z, V ! Y*g*. Let us compute UX$^+$, i.e., the closure of UX.

- Initially UX$^+$ = UX
- Then we have UX$^+$ = UVX (as U ! V and U UX)
- Then we have UX$^+$ = UVWX (as U ! W and U UVX)
- Then we have UX$^+$ = UVWXY (as WX ! Y and WX UVWX)
- Finally, we have UX$^+$ = UVWXYZ (as WX ! Z and WX UVWXY)

Note: The closure of UX covers all the attributes in R.

Decomposition of a relation

If a relation is not in a desired normal form, it can be decomposed into multiple relations such that each decomposed relation satisfies the required normal form.

Suppose a relation R consists of a set of attributes $A(R) = \{A_1, A_2, \ldots, A_n\}$. A *decomposition* of R replaces R by a set of (two or more) relations $\{R_1, \ldots, R_m\}$ such that both the following conditions hold:

- $\forall i : A(R_i) \subseteq A(R)$
- $A(R_1) \cup \cdots \cup A(R_m) = A(R)$

Decomposition criteria

The decomposition of a relation might aim to satisfy different criteria as listed below:

- Preservation of the same relation through join (lossless-join)
- Dependency preservation
- Repetition of information

Preservation of the same relation through join

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_2

.

&

X	Y
x_1	y_1
x_1	y_2

X	Z
x_1	z_1
x_1	z_2

X	Z
x_1	z_1
x_1	z_2

Y	Z
y_1	z_1
y_2	z_2

& .

& .

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_1	y_2	z_1
x_1	y_2	z_2

Lossy-join decomposition

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_2

Lossless-join decomposition

Preservation of the same relation through join

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_2

.

&

X	Y
x_1	y_1
x_1	y_2

X	Z
x_1	z_1
x_1	z_2

X	Z
x_1	z_1
x_1	z_2

Y	Z
y_1	z_1
y_2	z_2

& .

& .

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_1	y_2	z_1
x_1	y_2	z_2

Lossy-join decomposition

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_2

Lossless-join decomposition

Is the decomposition into <XY> and <YZ> lossy or lossless?

Testing for lossless-join decomposition

A decomposition of R into $\{R_1, R_2\}$ is *lossless-join*, iff $A(R_1) \cap A(R_2) \rightarrow A(R_1)$ or $A(R_1) \cap A(R_2) \rightarrow A(R_2)$ in F^+.

Consider the example of a relation R = <UVWXY> and the set of FDs = $\{$U $\rightarrow$ VW, WX $\rightarrow$ Y, V $\rightarrow$ X, Y $\rightarrow$ U$\}$.

Note that, the decomposition R_1 = <UVW> and R_2 = <WXY> is not lossless-join because $R_1 \cap R_2$ = W, and W is neither a key for R_1 nor for R_2.

However, the decomposition R_1 = <UVW> and R_2 = <UXY> is lossless-join because $R_1 \cap R_2$ = U, and U is a key for R_1.

Dependency preservation

The decomposition of a relation R with respect to a set of FDs F replaces R with a set of (two or more) relations $\{R_1, \ldots, R_m\}$ with FDs $\{F_1, \ldots, F_m\}$ such that F_i is the subset of dependencies in F^+ (the closure of F) that include only the attributes in R_i.

The decomposition is *dependency preserving* iff $(\bigcup_i F_i)^+ = F^+$.

Note: Through dependency preserving decomposition, we want to minimize the cost of global integrity constraints based on FDs' (i.e., avoid big joins in assertions).

Testing for dependency preserving decomposition

Consider the example of a relation R = <XYZ>, having the key X, and the set of FDs = *f*X *!* Y, Y *!* Z, X *!* Z*g*.

Note that, the decomposition R_1 = <XY> and R_2 = <XZ> is lossless-join but not dependency preserving because F_1 = *f*X *!* Y*g* and F_2 = *f*X *!* Z*g* incur the loss of the FD *f*Y *!* Z*g*, resulting into (F_1 *[* F_2)$^+$ 6= F^+.

However, the decomposition R_1 = <XY> and R_2 = <YZ> is lossless-join and also dependency preserving because F_1 = *fX ! Yg* and F_2 = *fY ! Zg*, satisfying (F_1 *[* F_2)$^+$ = F^+.

Third normal form

Definition (Third normal form (3NF))

A relational schema R is in 3NF if for every non-trivial functional dependency $X \rightarrow A$, where $X \cap A = \emptyset$, one of the following statements is true:

1. X is a superkey of R.
2. A is a part of any candidate key of R.

Note: A *superkey* is a set of one or more attributes that can uniquely identify an entity in the entity set.

Third normal form

The following relation is in 2NF but not in 3NF because `Country` is a non-prime attribute that depends on `Company`, which is again a non-prime attribute. Notably, the candidate key in this relation is *f*`PID`*g*.

PID	Company	Country	Make	Model	Distributor
P01	Maruti	India	WagonR	LXI	Carwala
P02	Maruti	India	WagonR	VXI	Carwala
P03	Maruti	India	Ertiga	VXI	Carwala
P04	Maruti	India	WagonR	LXI	Bhalla
P05	Honda	Japan	City	SV	Bhalla
P06	Tesla	USA	RAV4	EV	CarTrade
P07	Toyota	Japan	RAV4	EV	CarTrade
P08	BMW	Germany	X1	Expedition	CarTrade

We can convert this relation into 3NF!!!

Third normal form

PID	Company	Country	Make	Model	Distributor
P01	Maruti	India	WagonR	LXI	Carwala
P02	Maruti	India	WagonR	VXI	Carwala
P03	Maruti	India	Ertiga	VXI	Carwala
P04	Maruti	India	WagonR	LXI	Bhalla
P05	Honda	Japan	City	SV	Bhalla
P06	Tesla	USA	RAV4	EV	CarTrade
P07	Toyota	Japan	RAV4	EV	CarTrade
P08	BMW	Germany	X1	Expedition	CarTrade

- PID ! fCompany, Country, Make, Model, Distributorg
- fCompany, Make, Model, Distributorg ! Country (Violating 3NF)

Third normal form

Approach: Decompose the relation into multiple relations.

Company	Country
Maruti	India
Honda	Japan
Tesla	USA
Toyota	Japan
BMW	Germany

PID	Company	Make	Model	Distributor
P01	Maruti	WagonR	LXI	Carwala
P02	Maruti	WagonR	VXI	Carwala
P03	Maruti	Ertiga	VXI	Carwala
P04	Maruti	WagonR	LXI	Bhalla
P05	Honda	City	SV	Bhalla
P06	Tesla	RAV4	EV	CarTrade
P07	Toyota	RAV4	EV	CarTrade
P08	BMW	X1	Expedition	CarTrade

Note: Each non-trivial functional dependency in the left relation is on the superkey *f*`Company`*g*, and in the right relation is on the superkey *f*`PID`*g*.

Boyce-Codd normal form

Definition (Boyce-Codd normal form (BCNF))

A relational schema R is in BCNF if for every non-trivial functional dependency $X \rightarrow A$, where $X \cap A = \emptyset$, X is a superkey of R.

Note: A *superkey* is a set of one or more attributes that can uniquely identify an entity in the entity set.

Boyce-Codd normal form

The following relation is in 3NF but not in BCNF because the attribute `Distributor`, which depends on the non-key attribute `ShopID`, is a part of the key. Notably, the candidate key in this relation is *f*`Company`, `Make`, `Model`, `Distributor`*g*.

Company	Make	Model	Distributor	ShopID
Maruti	WagonR	LXI	Carwala	S1
Maruti	WagonR	VXI	Carwala	S1
Maruti	Ertiga	VXI	Carwala	S2
Maruti	WagonR	LXI	Bhalla	S3
Honda	City	SV	Bhalla	S4
Tesla	RAV4	EV	CarTrade	S5
Toyota	RAV4	EV	CarTrade	S5
BMW	X1	Expedition	CarTrade	S6
BMW	X1	Expedition	CarTrade	S6

We can convert this relation into BCNF!!!

Boyce-Codd normal form

Company	Make	Model	Distributor	ShopID
Maruti	WagonR	LXI	Carwala	S1
Maruti	WagonR	VXI	Carwala	S1
Maruti	Ertiga	VXI	Carwala	S2
Maruti	WagonR	LXI	Bhalla	S3
Honda	City	SV	Bhalla	S4
Tesla	RAV4	EV	CarTrade	S5
Toyota	RAV4	EV	CarTrade	S5
BMW	X1	Expedition	CarTrade	S6
BMW	X1	Expedition	CarTrade	S6

- $\{\texttt{Company}, \texttt{Make}, \texttt{Model}, \texttt{Distributor}\} \rightarrow \texttt{ShopID}$
- $\texttt{ShopID} \rightarrow \texttt{Distributor}$ (Violating BCNF)

Boyce-Codd normal form

Approach: Decompose the relation into multiple relations.

Distributor	ShopID
Carwala	S1
Carwala	S2
Bhalla	S3
Bhalla	S4
CarTrade	S5
CarTrade	S6

Company	Make	Model	ShopID
Maruti	WagonR	LXI	S1
Maruti	WagonR	VXI	S1
Maruti	Ertiga	VXI	S2
Maruti	WagonR	LXI	S3
Honda	City	SV	S4
Tesla	RAV4	EV	S5
Toyota	RAV4	EV	S5
BMW	X1	Expedition	S6

Note: Each non-trivial functional dependency in the left relation is on the superkey *f*`ShopID`*g*. What about the relation in the right?

Decomposition into BCNF – An algorithm

Result := *fRg* and *ag* := FALSE
Compute F^+
while NOT *ag* **do**
  if There is a schema R_i 2 *Result* that is not in BCNF **then**
    Let *X* ! *Y* be a non-trivial functional dependency that holds on R_i such that (*X* ! R_i) 2̸ F^+ and *X* \ *Y* = ϕ.
    Result := (*Result* R_i) [(R_i *Y*) [(*X*, *Y*) // This is simply decomposing *R* into *R* *Y* and *XY* provided *X* ! *Y* in *R* violates BCNF
  else
    ag := TRUE
  end if
end while

Decomposition into BCNF – An algorithm

Result := *fRg* and *ag* := FALSE
Compute F^+
while NOT *ag* **do**
  if There is a schema R_i *2 Result* that is not in BCNF **then**
    Let X ! Y be a non-trivial functional dependency that holds on R_i such that (X ! R_i) 2̸ F^+ and X \ $Y = \phi$.
    Result := (*Result* R_i) [(R_i Y) [(X, Y) // This is simply decomposing R into R Y and XY provided X ! Y in R violates BCNF
  else
    ag := TRUE
  end if
end while

<u>Note</u>: This decomposition process ensures lossless property.

Decomposition into BCNF – Example I

Consider a relation $R = <ABCDE>$ having the functional dependencies $\{A \rightarrow BC, C \rightarrow DE\}$.

Decomposition into BCNF – Example I

Consider a relation R **=** $<$**ABCDE**$>$ **having the functional dependencies** f**A** $!$ **BC, C** $!$ **DE**g**.**

Solution: The attribute closures provide $A^+ = ABCDE$, $B^+ = B$, $C^+ = CDE$, $D^+ = D$, and $E^+ = E$. Hence, A is the key of R.

Decomposition into BCNF – Example I

Consider a relation R = <ABCDE> having the functional dependencies $\{$A $\rightarrow$ BC, C $\rightarrow$ DE$\}$.

<u>**Solution**</u>**:** The attribute closures provide $A^+ = ABCDE$, $B^+ = B$, $C^+ = CDE$, $D^+ = D$, and $E^+ = E$. Hence, A is the key of R.

Note that, the functional dependency A $\rightarrow$ BC does not violate BCNF but C $\rightarrow$ DE does violate. By applying C $\rightarrow$ DE, we decompose R and obtain <ABC> and <CDE>.

Decomposition into BCNF – Example I

Consider a relation R = <ABCDE> having the functional dependencies $\{$A $\rightarrow$ BC, C $\rightarrow$ DE$\}$.

Solution: The attribute closures provide $A^+ = ABCDE$, $B^+ = B$, $C^+ = CDE$, $D^+ = D$, and $E^+ = E$. Hence, A is the key of R.

Note that, the functional dependency A $\rightarrow$ BC does not violate BCNF but C $\rightarrow$ DE does violate. By applying C $\rightarrow$ DE, we decompose R and obtain <ABC> and <CDE>.

Now both <ABC> (A is the key) and <CDE> are in BCNF (C is the key).

Decomposition into BCNF – Example II

Suppose a relation *R* = <ABCD> is given with the functional dependencies *f*AB ! C, B ! D, C ! A*g*.

Decomposition into BCNF – Example II

Suppose a relation R = <ABCD> is given with the functional dependencies *f*AB ! C, B ! D, C ! A*g*.

<u>Solution</u>: The attribute closures provide $A^+ = A$, $B^+ = BD$, $C^+ = AC$, $D^+ = D$, $AB^+ = ABCD$, and $BC^+ = ABCD$. Hence, AB and BC are the keys of R. Note that, the functional dependency AB ! C does not violate BCNF but B ! D and C ! A do violate. By applying B ! D, we decompose R and obtain <ABC> and <BD>.

Decomposition into BCNF – Example II

Suppose a relation *R* = <ABCD> is given with the functional dependencies *f*AB ! C, B ! D, C ! A*g*.

<u>Solution</u>: The attribute closures provide $A^+ = A$, $B^+ = BD$, $C^+ = AC$, $D^+ = D$, $AB^+ = ABCD$, and $BC^+ = ABCD$. Hence, AB and BC are the keys of *R*. Note that, the functional dependency AB ! C does not violate BCNF but B ! D and C ! A do violate. By applying B ! D, we decompose *R* and obtain <ABC> and <BD>.

Now <BD> is in BCNF (B is the key) but not <ABC>. The functional dependency C ! A violates BCNF. By applying C ! A, we further decompose <ABC> and obtain <BC> and <CA>. Now <BD>, <BC> and <CA> are all in BCNF.

Decomposition into BCNF – Example II

Suppose a relation R = <ABCD> is given with the functional dependencies fAB ! C, B ! D, C ! Ag.

Solution: The attribute closures provide A$^+$ = A, B$^+$ = BD, C$^+$ = AC, D$^+$ = D, AB$^+$ = ABCD, and BC$^+$ = ABCD. Hence, AB and BC are the keys of R. Note that, the functional dependency AB ! C does not violate BCNF but B ! D and C ! A do violate. By applying B ! D, we decompose R and obtain <ABC> and <BD>.

Now <BD> is in BCNF (B is the key) but not <ABC>. The functional dependency C ! A violates BCNF. By applying C ! A, we further decompose <ABC> and obtain <BC> and <CA>. Now <BD>, <BC> and <CA> are all in BCNF.

Note: This BCNF decomposition does not preserve dependencies.

Comments

Note that

- BCNF is stronger than 3NF – if a schema R is in BCNF then it is also in 3NF.
- 3NF is stronger than 2NF – if a schema R is in 3NF then it is also in 2NF.
- 2NF is stronger than 1NF – if a schema R is in 2NF then it is also in 1NF.

Comparison

Decomposition	**Preservation of the same relation through join**	**Functional dependency preservation**
1NF	lossless	kept
2NF	lossless	kept
3NF	lossless	kept
BCNF	lossless	lost

Elementary key normal form

Definition (Elementary key normal form (EKNF))

A relational schema R is in EKNF if for every non-trivial functional dependency $X \rightarrow A$, one of the following statements is true:

1. X is a superkey of R.
2. X is an elementary key attribute

Note: A non-trivial functional dependency $X \rightarrow Y$ is an elementary dependency if there exist no partial dependency. A key K is elementary key if $K \rightarrow Y$ is an elementary dependency.

Multi-valued dependency

Consider a relation schema R, and let X R and Y R. The functional dependency X Y holds on schema R if

$$t1[X] = t2[X],$$

in any legal relation $r(R)$, for all pairs of tuples $t1$ and $t2$ in r, implies

- $t1[X] = t2[X] = t3[X] = t4[X]$
- $t1[Y] = t3[Y]$ and $t2[Y] = t4[Y]$
- $t1[Z] = t4[Z]$ and $t2[Z] = t3[Z]$

where the two tuples $t3$ and $t4$ are also in r and Z denotes R $(X$ [$Y)$.

Multi-valued dependency

Consider a relation schema R, and let $X \quad R$ and $Y \quad R$. The functional dependency $X \quad Y$ holds on schema R if

$$t1[X] = t2[X],$$

in any legal relation $r(R)$, for all pairs of tuples $t1$ and $t2$ in r, implies

- $t1[X] = t2[X] = t3[X] = t4[X]$
- $t1[Y] = t3[Y]$ and $t2[Y] = t4[Y]$
- $t1[Z] = t4[Z]$ and $t2[Z] = t3[Z]$

where the two tuples $t3$ and $t4$ are also in r and Z denotes $R \quad (X \; [\; Y)$.

Note: The tuples $t1$, $t2$, $t3$ and $t4$ are not necessarily distinct.

Visualizing multi-valued dependency

	X	Y	$R \quad (X \;[\; Y)$
$t1$	$m_1 ... m_i$	$m_{i+1} ... m_j$	$m_{j+1} ... m_k$
$t2$	$m_1 ... m_i$	$n_{i+1} ... n_j$	$n_{j+1} ... n_k$

Visualizing multi-valued dependency

	X	Y	$R \quad (X\ [\ Y)$
$t1$	$m_1 ... m_i$	$m_{i+1} ... m_j$	$m_{j+1} ... m_k$
$t2$	$m_1 ... m_i$	$n_{i+1} ... n_i$	$n_{j+1} ... n_k$
$t3$	$m_1 ... m_i$	$m_{i+1} ... m_j$	$n_{j+1} ... n_k$
$t4$	$m_1 ... m_i$	$n_{i+1} ... n_i$	$m_{j+1} ... m_k$

Visualizing multi-valued dependency

	X	Y	R $(X$ [$Y)$
$t1$	$m_1 ... m_i$	$m_{i+1} ... m_j$	$m_{j+1} ... m_k$
$t2$	$m_1 ... m_i$	$n_{i+1} ... n_j$	$n_{j+1} ... n_k$
$t3$	$m_1 ... m_i$	$m_{i+1} ... m_j$	$n_{j+1} ... n_k$
$t4$	$m_1 ... m_i$	$n_{i+1} ... n_j$	$m_{j+1} ... m_k$

An example of X Y

Inference rules for multi-valued dependency

- If $X \twoheadrightarrow Y$ holds, then $X \twoheadrightarrow (R - (X \cup Y))$ holds.
- If $X \twoheadrightarrow Y$ holds and $W \supseteq Z$, then $WX \twoheadrightarrow YZ$ holds.
- If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$ both holds, then $X \twoheadrightarrow (Z - Y)$ holds.
- If $X \rightarrow Y$ holds, then $X \twoheadrightarrow Y$ holds.
- If $X \twoheadrightarrow Y$ holds and there exists W such that (a) $W \cap Y = \phi$, (b) $W \rightarrow Z$ and (c) $Y \supseteq Z$, then $X \rightarrow Z$ holds.

Fourth normal form

Definition (Fourth normal form (4NF))

A relational schema R is in 4NF if for every non-trivial multi-valued dependency $X \quad A$, X is a superkey of R.

Note: A *superkey* is a set of one or more attributes that can uniquely identify an entity in the entity set.

Fourth normal form

The following relation is not in 4NF because it satisfies the multi-valued dependency Name Age in which Name is not a superkey.

Name	Age	Codeword	Media
Irfan	28	abc	News
Irfan	40	xyz	Radio
Irfan	40	abc	News
Irfan	28	xyz	Radio
Imran	42	abc	News

We can convert this relation into 4NF!!!

Fourth normal form

Approach: Decompose the relation into multiple relations.

Name	Age
Irfan	28
Irfan	40
Imran	42

Name	Codeword	Media
Irfan	abc	News
Irfan	xyz	Radio
Imran	abc	News

Note: No multi-valued dependency exists in the decomposed relations.

Decomposition into 4NF – An algorithm

$Result$:= $\{R\}$ and ag := FALSE
Compute D^+ // Given schema R_i, let D_i denote the restriction of D^+ to R_i
while NOT ag **do**
  if There is a schema $R_i \in Result$ that is not in 4NF w.r.t. D_i **then**
    Let $X \quad Y$ be a non-trivial functional dependency that holds on R_i such that $(X \rightarrow R_i) \notin D_i$ and $X \cap Y = \phi$.
    $Result := (Result \quad R_i) \cup (R_i \quad Y) \cup (X, Y)$ // Decompose R into $R \quad Y$ and XY provided $X \quad Y$ in R violates 4NF
  else
    ag := TRUE
  end if
end while

Decomposition into 4NF – An algorithm

Result := *fRg* and *ag* := FALSE
Compute D^+ // Given schema R_i, let D_i denote the restriction of D^+ to R_i
while NOT *ag* **do**
  if There is a schema R_i *2 Result* that is not in 4NF w.r.t. D_i **then**
    Let X Y be a non-trivial functional dependency that holds on R_i such that (X ! R_i) $\not2$ D_i and X \ $Y = \phi$.
    Result := (*Result* R_i) [(R_i Y) [(X, Y) // Decompose R into R Y and XY provided X Y in R violates 4NF
  else
    ag := TRUE
  end if
end while

Note: The decomposition process ensures lossless property

Join dependency

Given a relation schema R, a join dependency $JD(R_1, R_2, \ldots, R_n)$ is defined by the constraint that every legal relation $r(R)$ should have a non-additive join decomposition into $R_1, R_2, \ldots, R_n$, i.e. for every such r we have

$$(\pi_{R_1}(r), \pi_{R_2}(r), \ldots, \pi_{R_n}(r)) = r.$$

Note: Multi-valued dependency is a special case of join dependency where $n = 2$.

Fifth normal form

Definition (Fifth normal form (5NF))

A relational schema R is in 5NF if for every non-trivial join dependency $JD(R_1, R_2, \ldots, R_n)$ in F^+, every R_i is a superkey of R.

Fifth normal form

Definition (Fifth normal form (5NF))

A relational schema R is in 5NF if for every non-trivial join dependency $JD(R_1, R_2, \ldots, R_n)$ in F^+, every R_i is a superkey of R.

Note: 5NF is also known as project-join normal form.

Domain key normal form

Definition (Domain key normal form (DKNF))

A relational schema R is in DKNF if all the constraints and dependencies that should hold on the valid relation states is a logical consequence of the domain and key constraints on the relation.

Sixth normal form

Definition (Sixth normal form (6NF))

A relational schema R is in 6NF if there exists no non-trivial join dependencies at all (with reference to generalized join operator).

References

- E. F. Codd (1970) *CACM*, 13(6):377-387.
- E. F. Codd (1971) *IBM Research Report*, RJ909.
- E. F. Codd (1974) *IBM Research Report*, RJ1385.
- R. Fagin (1977) *ACM TDS*, 2(3), 262-278.
- R. Fagin (1979) *IBM Research Report*, RJ2471.
- R. Fagin (1981) *CACM*, 6, 387-415.
- C. Zaniolo (1982) *ACM TDS*, 7(3), 489-499.
- C. J. Date (2002) *Temporal Data and the Relational Model*, Morgan Kaufmann.
- C. J. Date (2006) *Date on Database: Writings 2000-2006*, Springer-Verlag.