RELATIONAL DATABASES
3. Design theory for relational databases

Carmelo Vaccaro

University of Paris-Saclay

Master 1 - AI
2022/23: first semester

The content of these slides is taken from the book
Database Systems - The Complete Book,
by Hector Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom,
published by Pearson, 2014.

Contents of the chapter

1. Design of relational database schemas

Anomalies

In a database an *anomaly* is an unwanted feature of the database caused by bad design. The principal kinds of anomalies are:

1. **Redundancy.** Information repeated unnecessarily in several tuples.
2. **Update anomalies.** We may change information in one tuple but leave the same information unchanged in another.
3. **Deletion anomalies.** If a set of values becomes empty, we may lose other information as a side effect.

Redundancy

We have redundancy when information may be repeated unnecessarily in several tuples, like for example the length and genre for movies in the table below,

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure: The relation *Movies1*

Update anomalies

We may change information in one tuple but leave the same information unchanged in another.

For example, if we found that the movie *Star Wars* is really 125 minutes long, we might carelessly change the length in the first tuple of the table but not in the second or third tuples.

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure: The relation *Movies1*

One can argue that one should never be so careless, but it is possible to redesign the table so that the risk of such mistakes does not exist.

Deletion anomalies

If a set of values becomes empty, we may lose other information as a side effect.

For example, should we delete Vivien Leigh from the set of stars of *Gone With the Wind*, then we have no more stars for that movie in the database. The last tuple for *Gone With the Wind* in the table would disappear, and with it information that it is 231 minutes long and a drama.

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure: The relation *Movies1*

2. Decompositions of relations

Decomposing relations

In order to eliminate the anomalies of a relation R we can decompose the relation into two or more relations by splitting the set of attributes into subsets (not necessarily disjoint).

Decomposing relations: example 1/4

Example. We decompose the *Movies1* relation seen before into the following two relations *Movies2* and *Movies3*,

title	*year*	*length*	*genre*	*studioName*
Star Wars	1977	124	sciFi	Fox
Gone With the Wind	1939	231	drama	MGM
Wayne's World	1992	95	comedy	Paramount

Figure: The relation *Movies2*

title	*year*	*starName*
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Gone With the Wind	1939	Vivien Leigh
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Figure: The relation *Movies3*

Decomposing relations: example 2/4

The redundancy has been eliminated; for example, the length of each film appears only once, in relation *Movies2*.

The risk of an update anomaly is gone. For instance, since we only have to change the length of *Star Wars* in one tuple of *Movies2*, we cannot wind up with two different lengths for that movie.

title	*year*	*length*	*genre*	*studioName*
Star Wars	1977	124	sciFi	Fox
Gone With the Wind	1939	231	drama	MGM
Wayne's World	1992	95	comedy	Paramount

Figure: The relation *Movies2*

Decomposing relations: example 3/4

Also the risk of a deletion anomaly is gone. If we delete all the stars for *Gone With the Wind*, say, that deletion makes the movie disappear from *Movies3*. But all the other information about the movie can still be found in *Movies2*.

title	*year*	*starName*
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Gone With the Wind	1939	Vivien Leigh
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Figure: The relation *Movies3*

It might appear that *Movies3* still has redundancy, since the title and year of a movie can appear several times. However, these two attributes form a key for movies, and there is no more succinct way to represent a movie.

Decomposing relations: example 4/4

Moreover, *Movies3* does not offer an opportunity for an update anomaly.

If we changed to 2008 the year in the Carrie Fisher tuple, but not the other two tuples for *Star Wars*, then there would not be an update anomaly because there can be a different movie named *Star Wars* in 2008, and Carrie Fisher may star in that one as well.

title	*year*	*starName*
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Gone With the Wind	1939	Vivien Leigh
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Figure: The relation *Movies3*

Boyce-Codd normal form

Relations *Movies2* and *Movies3* are in the so called *Boyce-Codd normal form*, which means that they verify a special property implying that anomalies do not occur.

On the other side, *Movies1* is not in Boyce-Codd normal form, this is why it exhibits anomalies.

In order to define the Boyce-Codd normal form we need to introduce the concept of *functional dependencies* (in the next section).

Properties of a decomposition

We would like that a decomposition has three distinct properties:

1. Elimination of anomalies.
2. Recoverability of information: recovering the original relation from the tuples in its decomposition. This property is called *lossless join*.
3. Preservation of dependencies: ensure that by reconstructing the original relation from the decomposition by joining, the result will satisfy the original functional dependencies.

Example: a bad decomposition 1/2

We show an example of decomposition without the lossless join, that is by joining the decomposed relations we obtain a relation different from the original one.

Let us take the following relation $R(A, B, C)$

A	B	C
1	2	3
4	2	5

and let us decompose it into relations $R_1 = \pi_{A,B}(R)$ and $R_2 = \pi_{B,C}(R)$.

Table: R_1

A	B
1	2
4	2

Table: R_2

B	C
2	3
2	5

Example: a bad decomposition 2/2

When we try to reconstruct R by the natural join of the projected relations, we get the following relation, that we call S

A	B	C
1	2	3
1	2	5
4	2	3
4	2	5

that has two bogus tuples, (1, 2, 5) and (4, 2, 3), that were not in the original relation R.

Remark

The previous example tells us two things:

1. Not every decomposition is good, even one that is in BCNF (we will see later that every two-attribute relation is in BCNF).
2. There does not exist a way to obtain the original relation from its decompositions that works for every relation (and every decomposition). Indeed the relations R and S of the previous example have the same projections, so from them it is not possible to know if the original relation was R or S.

3. Functional dependencies

Definition of functional dependencies

Let R be a relation and let $A_1, A_2, \cdots, A_n, B_1, B_2, \cdots, B_m$ be attributes of R.

We say that $A_1, A_2, \cdots, A_n$ *functionally determine* $B_1, B_2, \cdots, B_m$ (denoted $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$) if any two tuples of R agreeing on all of $A_1, A_2, \cdots, A_n$ must also agree on all of $B_1, B_2, \cdots, B_m$.

In that case we say that $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ is a *functional dependency (FD) on* R.

Functional dependencies: remarks

In a functional dependency

$$A_1 A_2 \cdots A_n \to B_1 B_2 \cdots B_m$$

the orders of $A_1, A_2, \cdots, A_n$ and $B_1, B_2, \cdots, B_m$ does not count, because we are considering the sets $\{A_1, A_2, \cdots, A_n\}$ and $\{B_1, B_2, \cdots, B_m\}$ and not the lists.

The functional dependency $A_1 A_2 \cdots A_n \to A_1 A_2 \cdots A_n$, where the left and right side coincide, always holds and is called a *trivial functional dependency*.

Functional dependencies: example

The relation *Movies1* below verifies the following FD:

$$title\ year \rightarrow length\ genre\ studioName$$

but not the FD:

$$title\ year \rightarrow starName$$

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure: The relation *Movies1*

Exercise

Let $R(A, B, C, D, E)$ be the relation

R	A	B	C	D	E
	a	j	j	a	1
	b	b	m	c	2
	c	m	m	d	3
	d	n	s	c	2
	e	m	b	a	1
	f	j	c	d	3
	a	j	b	c	4
	c	m	n	a	1

and let

$$\mathcal{F} = \{C \to E,\ A \to C,\ A \to B,\ CD \to A,\ CD \to E,\ BD \to E,\ D \to E\}$$

be a set of functional dependencies hold.

Which functional dependencies among $\mathcal{F}$ hold for R?

Solution

$C \rightarrow E$: No. See 2nd and 3rd tuples.

$A \rightarrow C$: No. See 1st and 7th tuples.

$A \rightarrow B$: **Yes.**

$CD \rightarrow A$: **Yes, all different**

$CD \rightarrow E$: **Yes, all different**

$BD \rightarrow E$: **Yes (ok for the tuples 5 and 8).**

$D \rightarrow E$: No (ok when the value of D is a but not when the value of D is c on the tuples 2 et 7).

Keys of relations

We say a set of one or more attributes $\{A_1, A_2, \cdots, A_n\}$ is a key for a relation R if:

1. those attributes functionally determine all other attributes of the relation. That is, it is impossible for two distinct tuples of R to agree on all $A_1, A_2, \cdots, A_n$;
2. no proper subset of $\{A_1, A_2, \cdots, A_n\}$ functionally determines all other attributes of R, i.e., a key must be minimal.

Keys of relations: example

Attributes {*title*, *year*, *starName*} form a key for the relation *Movies1*:

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure: The relation *Movies1*

To prove that it is enough to say that they functionally determine all the other attributes and that no proper subset of {*title*, *year*, *starName*} has this property.

Exercise

Consider the relation below :

stud_id	*name*	*login*	*age*	*avg*
50000	Jean-Michel	jmi@u-psud	18	10.5
53412	Lucien	lucien@u-psud	19	17.5
53420	Paulette	paulette@polytech	18	8
53434	Ernest	ernest@iut	17	14.5
54412	Kaima	kaim@ens	19	13.5
43674	Domenicu	domenicu@math	20	5
43675	Lucien	lucien@math	18	10

1. Find those attributes that cannot be keys.
2. Find all the possible keys.

Solution

1. Find those attributes that cannot be keys.
   Answer. *name* and *age* cannot be keys because more than one tuple with the values Lucien and 18 respectively.

2. Find all the possible keys.
   Answer. {login}, {name, age}, {avg}.

Functional dependencies and primary keys

A relation can have more than one key and we can designate one of these keys as the *primary key*.

The choice of which key is a primary key is arbitrary and a primary key has no special role in the theory of functional dependencies.

Superkeys

A set of attributes that contains a key is called a *superkey*. Every key is a superkey, however, some superkeys are not (minimal) keys.

Every superkey satisfies the first condition of a key: it functionally determines all other attributes of the relation but not necessarily the second condition, minimality.

Sometimes what we have called in this course *superkey* and *key* are called *key* and *candidate key* respectively.

4. Boyce-Codd normal form

Definition of Boyce-Codd normal form

A relation R is in *Boyce-Codd normal form* (BCNF) if for every non-trivial functional dependency

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$$

in R we have that $\{A_1, A_2, \cdots, A_n\}$ is a superkey for R.

That is, the left side of every nontrivial functional dependency must be a superkey, that is it must contain a key.

Boyce-Codd normal form: example 1

Relation *Movies1* is not in BCNF because the following FD holds

$$title\ year \rightarrow length\ genre\ studioName$$

but {*title*, *year*} is not a superkey. Indeed {*title*, *year*, *starName*} is a key.

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure: The relation *Movies1*

Boyce-Codd normal form: example 2

Movies2 and *Movies3* are in BCNF.

title	*year*	*length*	*genre*	*studioName*
Star Wars	1977	124	sciFi	Fox
Gone With the Wind	1939	231	drama	MGM
Wayne's World	1992	95	comedy	Paramount

Figure: The relation *Movies2*

title	*year*	*starName*
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Gone With the Wind	1939	Vivien Leigh
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Figure: The relation *Movies3*

Exercise

Let $R(A, B)$ be a relation with two attributes. Prove that R is in BCNF.

Solution

Let $R(A, B)$ be a relation with two attributes. Prove that R is in BCNF.

Answer. Let us consider all the possible functional dependencies.

1. If R has no non-trivial functional dependencies, then it is trivially in BCNF.
2. If $A \to B$ holds, but $B \to A$ does not hold, then A is the only key and is on the left side of every non-trivial functional dependency.
3. If $B \to A$ holds, but $A \to B$ does not hold, this case is symmetric to case (2).
4. If $A \to B$ and $B \to A$ hold, then both A and B are keys and all the non-trivial functional dependencies have a key on the left side.

BCNF and anomalies

In a relation in BCNF there are no redundancies based on functional dependencies. This eliminates update and deletion anomalies caused by these kinds of redundancies.

Decomposition into BCNF

The following result (that we do not prove) holds: any relation R which is not in BCNF can be decomposed into two or more relations $S_1, \cdots, S_n$ that are in BCNF and has the property of *lossless join*, i.e., R is equal to the natural join of $S_1, \cdots, S_n$.

Remark 1. Decomposing R into $S_1, \cdots, S_n$ means that if A is the set of all attributes of R, there exist sets $B_1, \cdots, B_n$ such that $A = B_1 \cup \cdots \cup B_n$ and $S_i = \pi_{B_i}(R)$ for every $i = 1, \cdots, n$, that is S_i is the projection of R on the attributes of B_i.

Remark 2. The natural join is associative and commutative, thus the previous result says that in whatever order we do the natural joins of $S_1, \cdots, S_n$ we will obtain R.

Decomposition into BCNF: example

We have seen in a previous slide that the relations *Movies2* and *Movies3* are in BCNF.

It is easy to see that they are a decomposition of *Movies1* and that the natural join between *Movies2* and *Movies3* gives *Movies1*.

The splitting/combining rule for functional dependencies

We now introduce one concept of functional dependencies, that of the splitting/combining rule:
the functional dependency

$$A_1 \cdots A_n \rightarrow B_1 \cdots B_m \tag{1}$$

is equivalent to the set of functional dependencies

$$A_1 \cdots A_n \rightarrow B_i \tag{2}$$

for $i = 1, \cdots, m$.

This is obvious because if $A_1, \cdots, A_n$ functionally determine $B_1, \cdots, B_m$ then they determine each one of the B_i. On the other hand if they functionally determine each one of the B_i then they determine all the B_i and thus $B_1, \cdots, B_m$.

The transitive rule

The transitive rule says that if in a relation R the following functional dependencies hold, $A_1 \cdots A_n \rightarrow B_1 \cdots B_m$ and $B_1 \cdots B_m \rightarrow C_1 \cdots C_k$, then also the functional dependency $A_1 \cdots A_n \rightarrow C_1 \cdots C_k$ holds.

Functional dependencies closure of a set of attributes

In order to introduce the procedure to decompose a relation into relations in BCNF we need to introduce the concept of *functional dependencies closure* of set a of attributes.

Suppose to have a relation R that has attributes $A_1, \cdots, A_n$. The functional dependencies closure of $\{A_1, \cdots, A_n\}$ is the set of attributes B such that $A_1 \cdots A_n \rightarrow B$ is a functional dependency in R.

Functional dependencies closure of a set of attributes: remarks

Remark 1. The functional dependencies closure of $\{A_1, \cdots, A_n\}$ is denoted $\{A_1, \cdots, A_n\}^+$.

Remark 2. We have that $\{A_1, \cdots, A_n\} \subset \{A_1, \cdots, A_n\}^+$ because for every $i = 1, \cdots, n$ it is trivial that $A_1 \cdots A_n \rightarrow A_i$ holds.

How to compute the functional dependencies closure of a set of attributes 1/3

The strategy to build $\{A_1, \cdots, A_n\}^+$ is the following: we start with $\{A_1, \cdots, A_n\}$ which necessarily is a subset of $\{A_1, \cdots, A_n\}^+$.

Then we add one by one an attribute that we have verified belongs to $\{A_1, \cdots, A_n\}^+$. When we cannot add anymore attributes we have finished to construct $\{A_1, \cdots, A_n\}^+$.

How to compute the functional dependencies closure of a set of attributes 2/3

The procedure is the following. Let us set $\mathcal{A}_0 := \{A_1, \cdots, A_n\}$. If for every attribute B not belonging to $\mathcal{A}_0$ the functional dependency

$$A_1 \cdots A_n \to B$$

does not hold then we can conclude that $\mathcal{A}_0$ is the closure. Otherwise let B be an attribute not belonging to $\mathcal{A}_0$ such that the functional dependency

$$A_1 \cdots A_n \to B$$

holds. Then B belongs to the closure and we set $\mathcal{A}_1 := \mathcal{A}_0 \cup \{B\}$. Now we do on $\mathcal{A}_1$ the same procedure seen on $\mathcal{A}_0$.

How to compute the functional dependencies closure of a set of attributes 3/3

In particular either we will conclude that $\mathcal{A}_1$ is the closure or we will have a new set $\mathcal{A}_2$ equal to $\mathcal{A}_1$ plus another attribute. In the latter case we do on $\mathcal{A}_2$ the same procedure seen on $\mathcal{A}_1$ and so on.

Since the number of attributes of R is finite there will exist a natural number m such that $\mathcal{A}_m$ is the closure.

Computing the functional dependencies closure of a set of attributes: example 1/2

Let us consider a relation R with attributes $\{A, B, C, D, E, F\}$. Suppose that R has the functional dependencies $AB \to D$, $BD \to AC$ and $BF \to E$. Let us calculate the closure of $\{A, B\}$.

By the splitting rule we have that $BD \to AC$ is equivalent to the two functional dependencies $BD \to A$ and $BD \to C$. Let us set $\mathcal{A}_0 := \{A, B\}$. Let us take the attribute C not belonging to $\mathcal{A}_0$. The only elements in $\mathcal{A}_0$ are A and B and $AB \to C$ does not hold, therefore we have to check another element not belonging to $\mathcal{A}_0$, for example D.

Computing the functional dependencies closure of a set of attributes: example 2/2

Since $\{A, B\}$ belongs to $\mathcal{A}_0$ and $AB \rightarrow D$ holds then D belongs to the closure, so we set $\mathcal{A}_1 := \mathcal{A}_0 \cup \{D\} = \{A, B, D\}$. Let us take the attribute C not belonging to $\mathcal{A}_1$. Since $BD \rightarrow C$ holds, then C belongs to the closure, so we set $\mathcal{A}_2 := \mathcal{A}_1 \cup \{C\} = \{A, B, C, D\}$.

Now let us take the attribute E not belonging to $\mathcal{A}_2$. The only functional dependency where E is on the right side is $BF \rightarrow E$ but F does not belong to $\mathcal{A}_2$. Finally if we take the attribute F not belonging to $\mathcal{A}_2$ we have that there are not functional dependency where F is on the right side.

Since there are no more attributes to consider, we conclude that the closure of $\{A, B\}$ is $\mathcal{A}_2 = \{A, B, C, D\}$.

Exercise

For the relation $S(A, B, C, D)$ with the functional dependencies $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow D$ compute the closures of all subsets of attributes. Find all the keys.

Solution

For the relation $S(A, B, C, D)$ with the functional dependencies $A \to B$, $B \to C$, $B \to D$ compute the closures of all subsets of attributes. Find all the keys.

Answer. We have that
$\{A\}^+ = \{A, B, C, D\}$
$\{B\}^+ = \{B, C, D\}$
$\{C\}^+ = \{C\}$
$\{D\}^+ = \{D\}$
The closures of the sets of order 2 or 3 containing A is $\{A, B, C, D\}$
$\{B, C\}^+ = \{B, C, D\}$
$\{B, D\}^+ = \{B, D, C\}$
$\{C, D\}^+ = \{C, D\}$
$\{B, C, D\}^+ = \{B, C, D\}$
The only subsets of attributes whose closure is all the attributes are those containing A. Among them the only that is minimal is $\{A\}$, which is the only key.

BCNF decomposition algorithm 1/2

The algorithm to decompose a relation into relations in BCNF is the following.

Let R be a relation and let $\mathcal{A}$ be the set of its attributes. If R is in BCNF then nothing needs to be done and the algorithm stops. Otherwise there is a BCNF violation, that is there is a non-trivial functional dependency $X \to Y$ with X not a superkey.

In that case we split $\mathcal{A}$ into the two subsets X^+ and $X \cup (\mathcal{A} \setminus X^+)$ and let R_1 and R_2 be the projections of R by these two sets of attributes respectively.

BCNF decomposition algorithm 2/2

We have that X^+ is strictly smaller than $\mathcal{A}$ since otherwise X would be a superkey. We have also that X is strictly smaller than X^+ because $X^+ \supset Y$ and $X \to Y$ is not trivial; this implies that $X \cup (\mathcal{A} \setminus X^+)$ is strictly smaller than $\mathcal{A}$.

Now we apply the algorithm to R_1 and R_2. The algorithm eventually stops because at each step the two sets of attributes obtained by splitting are strictly smaller than the original one and this process cannot continue indefinitely because $\mathcal{A}$ is finite.

BCNF decomposition algorithm: example 1

In the *Movies1* relation we have the BCNF violation

$$title\ year \rightarrow length\ genre\ studioName.$$

Let us set $X := \{title, year\}$; then

$$X^+ = \{title, year, length, genre, studioName\}.$$

The set of all attributes of *Movies1* is

$$\mathcal{A} := \{title, year, length, genre, studioName, starName\}.$$

By applying the algorithm we have the decomposition of $\mathcal{A}$ into X^+ and $X \cup (\mathcal{A} \setminus X^+) = \{title, year, starName\}$.

This decomposes *Movies1* into *Movies2* and *Movies3*, which is the required BCNF decomposition.

BCNF decomposition algorithm: example 2 1/2

Consider a relation with schema *(title, year, studioName, president, presAddr)*. The following three functional dependencies hold

$title\ year \rightarrow studioName,\ studioName \rightarrow president,\ president \rightarrow presAddr.$

The only key is $\{title, year\}$, thus the last two functional dependencies violate the BCNF.

By using the FD $studioName \rightarrow president$ we obtain the decomposition $\{studioName, president, presAddr\}$, $\{studioName, title, year\}$, where the first set is the closure of $\{studioName\}$.

BCNF decomposition algorithm: example 2 2/2

The relation obtained from the second set is in BCNF, while that from the first set has $\{studioName\}$ as the only key but has the FD $president \rightarrow presAddr$, thus it is not in BCNF.

By decomposing it using this FD we obtain the two sets of attributes $\{studioName, president\}$ and $\{president, presAddr\}$, that are in BCNF.

Exercise

We have seen previously that the the relation $S(A, B, C, D)$ with the functional dependencies $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow D$ has only one key $\{A\}$.

Prove that S is not in BCNF and apply the algorithm to decompose it into BCNF relations.

Solution

$B \to D$ is a BCNF violation. We set $X := \{B\}$, we have seen that $\{B\}^+ = \{B, C, D\}$, thus $X^+ = \{B, C, D\}$, therefore $\mathcal{A} \setminus X^+ = \{A\}$ and $X \cup (\mathcal{A} \setminus X^+) = \{A, B\}$.

Let $R_1(B, C, D)$ and $R_2(A, B)$ be the projections of R onto the given attributes. R_2 is in BCNF because it has two attributes. The only key of R_1 is B and the only functional dependencies that hold in it are $B \to C$ and $B \to D$, so there is no BCNF violation and R_1 too is in BCNF.

Properties of the BCNF decomposition algorithm

We have seen that a desirable decomposition of a relation should have the following three properties: 1) elimination of anomalies; 2) lossless join; 3) preservation of dependencies.

A relation in BCNF verifies the first property. The decomposition into BCNF seen at the slides 52 and 53 verifies the second property. Does it verify also the third?

The answer in general is no, that is there are relations whose decomposition according to the algorithm of slides 52 and 53 does not verify property 3. We see an example in the next slides.

Example 1/4

In the 1970's in the USA the distributors of a given city of arcade and pinball games wanted exclusivity deals from the producer. This meant that a video game, for example Space Race produced by the video game producer Atari could be distributed in the city of Chicago by only one distributor, for example Empire Distributing.

We model this situation with the following relation called *VG_distribution* that has the following attributes:

1. title, the name of a the video game
2. distributor, the name of the video game and pinball game distributor.
3. city, the city where the distributor is located.

Example 2/4

So we have the following two functional dependencies:

$$distributor \rightarrow city$$

$$title\ city \rightarrow distributor$$

where the first one follows from the fact that a distributor determines the city where it is located and the second one follows from the exclusivity deals described above.

Example 3/4

The only keys are $\{title, city\}$ and $\{title, distributor\}$ so $distributor \rightarrow city$ is a BCNF violation.

By applying the algorithm seen before we decompose the relation into BCNF relations $\{distributor, city\}$ and $\{title, distributor\}$. The first relation has *distributor* as key and the functional dependency $distributor \rightarrow city$ holds. The second relation has $\{title, distributor\}$ as key and has no non-trivial functional dependencies.

Example 4/4

Now the two relations

Table: R_1

distributor	*city*
Empire Distributing	Chicago
Fun Distributions	Chicago

Table: R_2

title	*distributor*
Space Race	Empire Distributing
Space Race	Fun Distributions

are permissible according to the functional dependencies that apply to each of the above relations, but when we join them we get two tuples

title	*distributor*	*city*
Space Race	Empire Distributing	Chicago
Space Race	Fun Distributions	Chicago

that violates the functional dependency *title city* $\rightarrow$ *distributor*.

How to solve this issue

The solution to the problem illustrated in the previous example is to relax the BCNF requirement slightly, in order to allow the occasional relation schema that cannot be decomposed into BCNF relations without our losing the ability to check the FD's.

This relaxed condition is called "third normal form" and is the topic of the next section.

5. Third normal form

Definition of third normal form

A relation R is in *third normal form* (3NF) if for every non-trivial functional dependency

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$$

in R we have that

- either $\{A_1, A_2, \cdots, A_n\}$ is a superkey for R
- or those of $B_1, B_2, \cdots, B_m$ that are not among the A's, are each a member of some key (not necessarily the same key).

BCNF implies third normal form

By definition a relation in BCNF is also in third normal form.

Prime attributes and third normal form

An attribute of a relation that is a member of some key is said to be *prime*.

Thus, the 3NF condition can be stated as "for each nontrivial FD, either the left side is a superkey, or the right side consists of prime attributes only".

Example

The relation *VG_distribution(title, distributor, city)* shown in the previous section is an example of a relation in 3NF but not in BCNF.

Indeed the only keys of this relation are $\{title, city\}$ and $\{title, distributor\}$ and the non-trivial functional dependencies are $distributor \rightarrow city$ and $title\ city \rightarrow distributor$.

The first of these functional dependencies is a BCNF violation, but it is not a 3NF violation because *city* is a prime attribute (it belongs to the key $\{title, city\}$).

We remark that in this relation all the attributes are prime. A relation where all attributes are prime is necessarily in 3NF.

Exercise

We have seen previously that the the relation $S(A, B, C, D)$ with the functional dependencies $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow D$ has only one key $\{A\}$.

Determine whether S is in 3NF or not.

Solution

The only prime attribute is A, thus $B \rightarrow C$ and $B \rightarrow D$ are 3NF violations, because B is not a key. Therefore S is not in 3NF.

Exercise

Let $R(A, B, C, D)$ be a relation with the functional dependencies $B \rightarrow C$, $B \rightarrow D$.

Determine whether R is in 3NF or not.

Solution

We must first find all the keys of R and for that we compute the closures of all subsets of attributes. All non-trivial functional dependencies have B on the left, so the closure of any subset of attributes not containing B is trivial.
$\{B\}^+ = \{B, C, D\}$
$\{A, B\}^+ = \{A, B, C, D\}$
$\{B, C\}^+ = \{B, C, D\}$
$\{B, D\}^+ = \{B, C, D\}$
The closures of the sets of order 3 containing A and B is $\{A, B, C, D\}$.
$\{B, C, D\}^+ = \{B, C, D\}$
The only key is $\{A, B\}$, the prime attributes are A and B and $B \to D$ is a 3NF violation because B is not a superkey and D is not prime.

The end.