Functional Dependencies

A Fresh Look

By Bill Rishel

Functional dependency analysis is an important tool in database design. The previous article in this series looked at a particular problem using functional dependencies from Joe Celko's book *SQL for Smarties.* We saw there that of the five solutions to a functional dependency problem provided by Celko, *four were incorrect*! Obviously functional dependency analysis is not so simple if even Joe Celko can make errors.

Here we dive into the subject of functional dependencies from the ground up.

For a given object x and set of attributes A, B, C, there are eight possible assignments of the attributes A, B, C to x:[1]

1. A(x), B(x), C(x)
2. A(x), B(x), ~C(x)
3. A(x), ~B(x), C(x)
4. ~A(x), B(x), C(x)
5. A(x), ~B(x), ~C(x)
6. ~A(x), B(x), ~C(x)
7. ~A(x), ~B(x), C(x)
8. ~A(x), ~B(x), ~C(x)

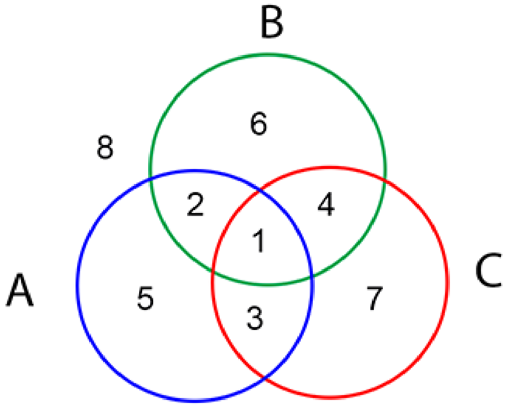


Figure 1

The the three attributes A, B, C, define an "attribute space", with eight distinct locations or regions in the space. Each number in Figure 1 corresponds to one combination of attributes as listed. Hence, each combination of the three attributes A, B, C designates one *location* in the attribute space.[2]

[1] If A is an attribute, ~A means not-A. For a given object x, if A applies to x we can write A(x). If A does not apply to x, we write ~A(x), which means "not A(x)".

[2] Figure 1 is an example of a *Venn Diagram.*

We can indicate that a particular region in the diagram is *empty* – meaning no object can have that combination of attributes – by "shading" that region in the diagram. For example, given attributes A, B, C, the functional dependency

$$A\ B \longrightarrow C$$

means (*A and B) implies C*. Logically, that is the same as saying:

not [A and B and ~C] [3]

which means the region in the diagram corresponding to A and B and ~C is empty. We can represent this in the diagram by shading the region (A and B and ~C) to indicate it is *empty*.

The blue shading in the diagram indicates that region is empty, meaning you cannot have an object with attributes A and B and ~C.

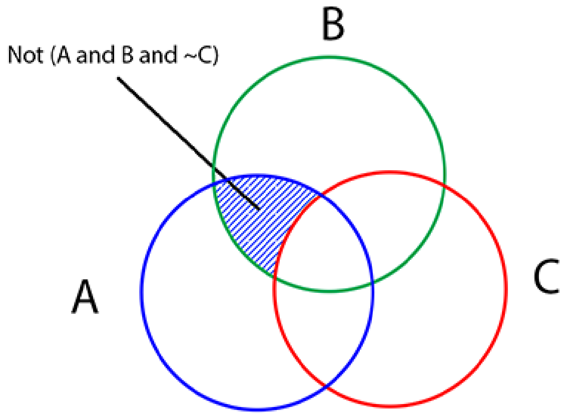


Figure 2

All functional dependencies represent *constraints* on allowable values for the attributes. In the example above, the combination (A and B and ~C) is *not allowed.* By converting a functional dependency from its implication form to its negation form – as we did in converting [A and B ⟶ C] to [not (A and B and ~C)] – we can more easily see in a diagram what the functional dependency is saying. As we shall discover, this is a very useful technique.

[3] In logic "X implies Y" is the same as "not (X and not Y)". This can seem counter-intuitive at first but is correct. In ordinary language: *If X implies Y, then if Y is false, X cannot be true. So either Y is true or X is false.*

Functional dependencies ordinarily are in *sets* of functional dependencies that are meant to be applied *simultaneously*. We next take a look at how the above diagramming technique works with a set of simultaneous functional dependencies.

A Simple Set of Functional Dependencies

A, B → D

A, C → B

B, C → A

Converting these functional dependencies from the implication form to the negation form we get:

not (A and B and ~D)
not (A and C and ~B)
not (B and C and ~A)

We can put these negation forms into a diagram and see how they look applied simultaneously in the same diagram. We use different kinds of shading for different functional dependencies.

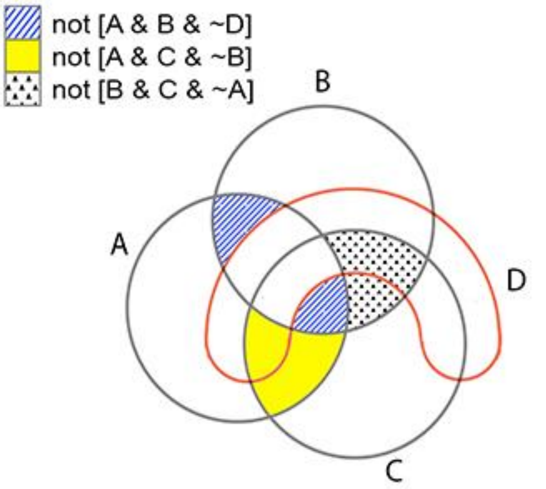


Figure 3

Keep in mind that the functional dependencies are *constraints* on what attribute combinations are allowed The three functional dependencies together require that all of the shown shaded regions in Figure 3 are empty. No object can take on the combination of attribute values indicated by *any* of the shaded regions in the diagram.

Functional Dependency Covers

The combination, or union, of the empty regions defined by all of the functional dependencies taken together is called the *cover* of the set of functional dependencies. Figure 4 below shows the combined cover for all the functional dependencies depicted in Figure 3 above.

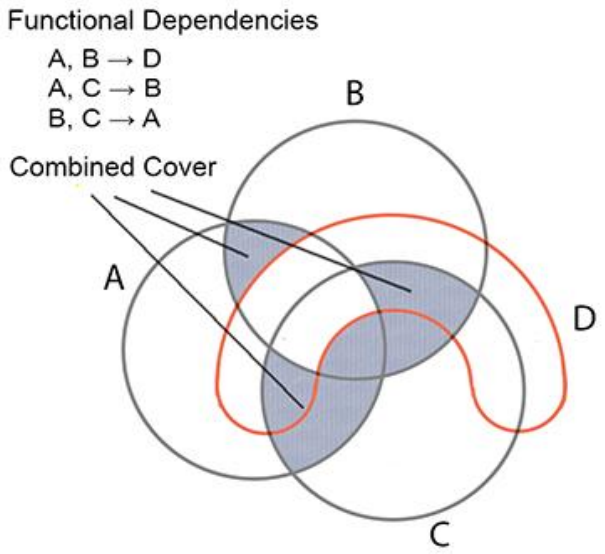


Figure 4

It is possible for another different set of functional dependencies to combine to form the *same cover* as the set given above. If so, the *combined cover, as* shaded in the diagram, would be the same, even though individual functional dependencies may have different shaded regions.

Consider this set of functional dependencies:

1. A, B → C

2. A, C → B

3. B, C → A

4. A, B → D

5. A, C → D

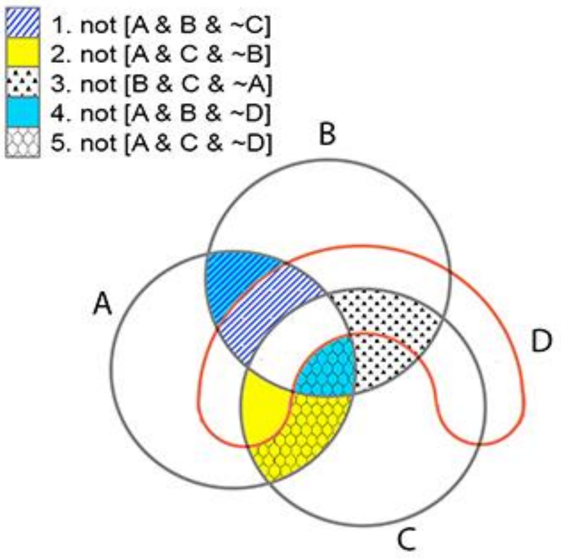


Figure 5

The regions for each functional dependency are shown in Figure 5 to the right.

The diagram in Figure 6 shows in gray the *combined* “cover” of all the functional dependencies. Are there any functional dependencies that could be removed from the set and still have the same combined cover?

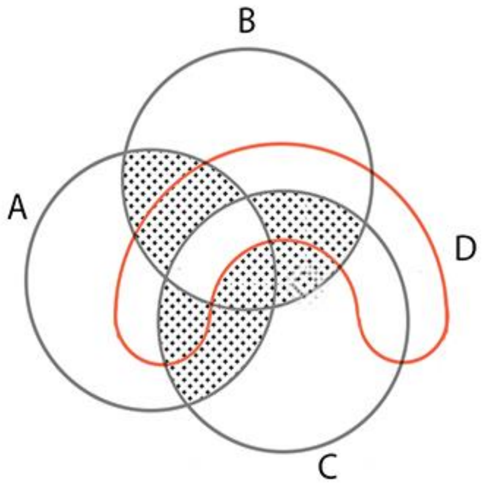


Figure 6

Exercise 1.

Give two different subsets of the above functional dependences with the same cover as the entire set.

[Hint: Note which regions in Figure 5 are shaded for more than one FD.]

Minimum Cover of a Set of Functional Dependencies

A *minimum cover* of a set of function dependencies means a *smallest* subset that has the same cover as the entire set.

Example

A, B → C

A, D → B

B, D → C

B, C → A

A, B → D

A, C → D

First convert the functional dependencies from implication form to negation form to see how the regions in the diagram are filled in:

1. A, B → C ⇒ not [A & B & ~C]
2. A, D → B ⇒ not [A & D & ~B]
3. B, D → C ⇒ not [B & D & ~C]
4. B, C → A ⇒ not [B & C & ~A]
5. A, B → D ⇒ not [A & B & ~D]
6. A, C → D ⇒ not [A & C & ~D]

A minimum cover is a subset of the given function dependences [1-6 above] that provides the same combined cover as the entire set (see Figure 8).

In this case there are two different minimum subsets that provide complete covers.

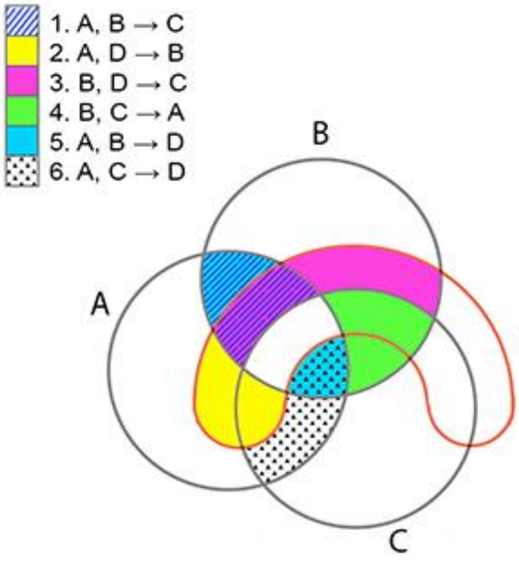


Figure 7

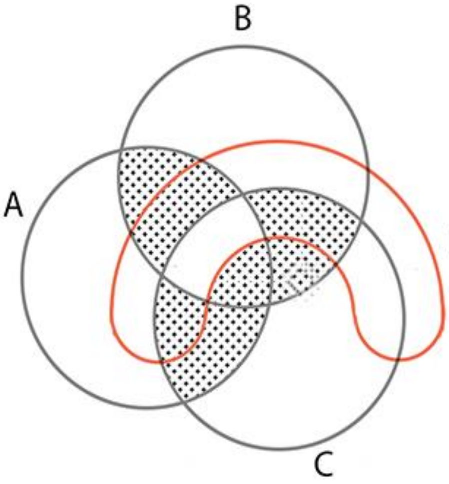


Figure 8

Exercise 2.

Give two minimum covers for the six functional dependencies above.

Attributes Spaces

Functional dependencies express relationships between attributes. A functional dependency is of the form: $\alpha_1, ..., \alpha_j \rightarrow \beta$, where $\alpha_1, ..., \alpha_j$ and β are attributes. The functional dependency $\alpha_1, ..., \alpha_j \rightarrow \beta$ expresses the relationship that if the values of attributes $\alpha_1, ..., \alpha_j$ are *known* then the value of attribute β is *known*.

The FD,

$$\alpha_1, ..., \alpha_j \rightarrow \beta$$

has the logical form,

α_1 and ... α_j *imply* β

which is equivalent to the logical statement,

not [α_1 and ... α_j and not-β]

or,

not [α_1 and ...and α_j and ~β]

Example

The FD,

$$A, B \rightarrow C$$

is equivalent to the logical statement,

not [A and B and ~C]

We say the functional dependency $A, B \rightarrow C$ *forbids* or *disallows* the condition,

A and B and ~C

Given a complete set of attributes referenced by a set of functional dependencies, each attribute can have a positive or negative form, so for k attributes there are 2^k different combinations of the k attributes.

For example, the three attributes A, B, C, can have eight different combinations,

A, B, C
~A, B, C
A, ~B, C
A, B, ~C
~A, ~B, C
~A, B, ~C
A, ~B, ~C
~A, ~B, ~C

Attribute Space of Four Attributes

0	A	B	C	D
1	~A	B	C	D
2	A	~B	C	D
3	~A	~B	C	D
4	A	B	~C	D
5	~A	B	~C	D
6	A	~B	~C	D
7	~A	~B	~C	D
8	A	B	C	~D
9	~A	B	C	~D
10	A	~B	C	~D
11	~A	~B	C	~D
12	A	B	~C	~D
13	~A	B	~C	~D
14	A	~B	~C	~D
15	~A	~B	~C	~D

A and B and ~C

An attribute space of k attributes has 2^k distinct locations in the space. For example an attribute space of four attributes has $2^4 = 16$ locations. The table at left represents an attribute space with four attributes.
The numbers in the left-most column correspond to the sixteen locations in the attribute space. The attribute values for a location designate for each attribute whether that attribute value is known or not known. In our notation, ~A indicates the attribute A is not known, while A indicates the attribute is known.

Example

A location with the attribute values ~A, B, C, D indicates the condition where B, C, D are known, amd A is not known.

The functional dependency A, B → C has the form of an implication: *A and B imply C*, which is equivalent to:

not [A and B and ~C]

The implication is actually a negative statement in that it asserts what *cannot occur*. The functional dependency is asserting that the condition:

A and B and ~C

cannot occur.

In terms of the attribute space represented by the table to the left, the functional dependency A, B → C asserts that any location (row) with A and B and ~C cannot occur. Note that two locations in the table are designated by A, B → C as forbidden or not allowed: 4 and 12. [colored blue]

Two FDs in the Same Attribute Space

0	A	B	C	D
1	~A	B	C	D
2	A	~B	C	D
3	~A	~B	C	D
4	A	B	~C	D
5	~A	B	~C	D
6	A	~B	~C	D
7	~A	~B	~C	D
8	A	B	C	~D
9	~A	B	C	~D
10	A	~B	C	~D
11	~A	~B	C	~D
12	A	B	~C	~D
13	~A	B	~C	~D
14	A	~B	~C	~D
15	~A	~B	~C	~D

As another example with the same attribute space, consider the functional dependency:

A → B

A → B means *not [A and ~B]*, so A → B disallows all locations in the attribute space with A and ~B. There are four locations in the attribute space that qualify: 2, 6, 10, and 14. [colored yellow]

We can represent more than one functional dependency in the same attribute space. For example, let's represent, in addition to A → B, the functional dependency:

B, C → D

B, C → D means *not [B and C and ~D]*, so B, C → D designates locations (A, B, C, ~D) and (~A, B, C, ~D) as forbidden, which are numbered 8 and 9. [colored red]

An FD Implied by Other FDs

0	A	B	C	D
1	~A	B	C	D
2	A	~B	C	D
3	~A	~B	C	D
4	A	B	~C	D
5	~A	B	~C	D
6	A	~B	~C	D
7	~A	~B	~C	D
8	A	B	C	~D
9	~A	B	C	~D
10	A	~B	C	~D
11	~A	~B	C	~D
12	A	B	~C	~D
13	~A	B	~C	~D
14	A	~B	~C	~D
15	~A	~B	~C	~D

Table 1

A functional dependency can be *implied by other functional dependencies*. *The essential meaning of a functional dependency* is that it forbids certain combinations of attribute values in the attribute space. Consequently, an FD is *implied* by a set of FDs if the set of FDs *together* forbids all the locations forbidden by the first FD. You might say the set of FDs "does the work of" the FD they imply.

Example

Consider the FD: A, C → B

A, C → B means that locations in the attribute space with A, C and ~B are forbidden. That corresponds to locations numbered 2 and 10 in the table at left. [gray shaded]

Now let's look for two FDs that together forbid the locations 2 and 10 in the attribute space.

Considerthe FD: A, D → B.

A, D → B means the locations with *A and D and ~B* are forbidden. Note that locations 2 and 6 have A and D and ~B [colored yellow].

Now consider the FD: A, C → D.

A, C → D means the locations with *A and C and ~D* are forbidden. That's locations 8 and 10 [colored red].

Notice that the two FDs A, D → B and A, C → D together forbid locations 2, 6, 8, and 10, which *include* the locations 2 and 10, disallowed by FD A, C → B. This means the two FDs

A, D → B and A, C → D *together imply* the FD A, C → B.

Using '⇒' for implies, we write:

A, D → B and A, C → D ⇒ A, C → B

Another Way to Represent an Attribute Space

	A	B	C	D
0	1	1	1	1
1	0	1	1	1
2	1	0	1	1
3	0	0	1	1
4	1	1	0	1
5	0	1	0	1
6	1	0	0	1
7	0	0	0	1
8	1	1	1	0
9	0	1	1	0
10	1	0	1	0
11	0	0	1	0
12	1	1	0	0
13	0	1	0	0
14	1	0	0	0
15	0	0	0	0

Another way to represent an attribute space is by using 0's and 1's in the attribute columns, indicating an attribute is not known with a 0, and indicating an attribute is known with a 1.

In the table at left row 4 indicates: A and B and ~C and D, and row 12 indicates A and B and ~C and ~D. [colored blue]

A functional dependency always has some number of attributes to the left of the arrow, and *one* attribute to the right. The negation form of an FD is always of the form:

not [$\alpha_1 \ldots \alpha_j$ and $\sim\beta$]

where the $\alpha_1 \ldots \alpha_j$ are always positive attributes, and the single $\sim\beta$ the only negative attribute.

Example

A, C → B forbids the locations with A and C and ~B, which is all the rows with a 1 in the A column, a 1 in the C column, and a 0 in the B column. [colored red]

Example

C, D → A forbids the locations with C and D and ~A (C=1, D=1, A=0) [colored yellow]

Representing an Attribute Space with 0's and 1's

	A	B	C	D	E
0	1	1	1	1	1
1	0	1	1	1	1
2	1	0	1	1	1
3	0	0	1	1	1
4	1	1	0	1	1
5	0	1	0	1	1
6	1	0	0	1	1
7	0	0	0	1	1
8	1	1	1	0	1
9	0	1	1	0	1
10	1	0	1	0	1
11	0	0	1	0	1
12	1	1	0	0	1
13	0	1	0	0	1
14	1	0	0	0	1
15	0	0	0	0	1
16	1	1	1	1	0
17	0	1	1	1	0
18	1	0	1	1	0
19	0	0	1	1	0
20	1	1	0	1	0
21	0	1	0	1	0
22	1	0	0	1	0
23	0	0	0	1	0
24	1	1	1	0	0
25	0	1	1	0	0
26	1	0	1	0	0
27	0	0	1	0	0
28	1	1	0	0	0
29	0	1	0	0	0
30	1	0	0	0	0
31	0	0	0	0	0

Representing an attribute space using 0's and 1's makes some patterns more evident. Previously we illustrated an FD that is implied by two other FDs because together they disallow all the locations in the attribute space disallowed by the FD they imply.

Let's look at A, C → B with a 0's and 1's representation of the attribute space. A, C → B forbids attribute combinations of A, C, and ~B, which are the locations with A=1, C=1, and B=0 in the table at left. [colored red]

Note that the attributes D and E are not included in the FD A, C → B. Looking at the combinations forbidden by A, C→B (in red), there are four possible combinations for D and E: 11, 01, 10, and 00. [colored yellow]

In the case where D and E are both 1 (row 2), only an FD with B to the right of the arrow can rule out that row.

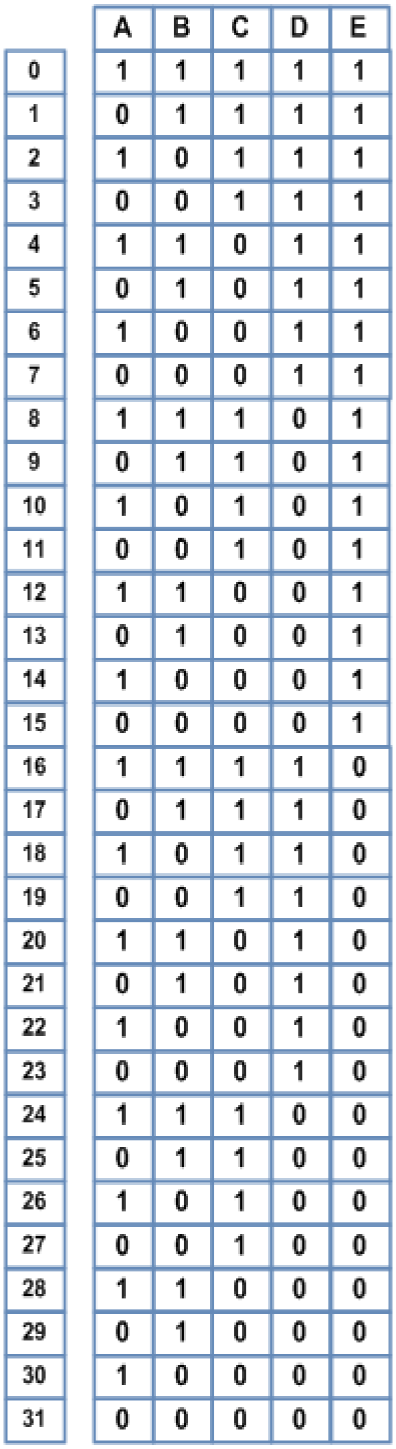
Can you see why? [1]

That tells us that any set of FDs that imply A, C → B must include an FD that has B to the right of the arrow.

Now consider the locations where the attributes not mentioned in A, C → B, again D and E, both have 0's, which is row 26. Since the only attributes with a 1 for this location are A and C, any FD that rules out this location can only have A and/or C on the left side of the arrow.

These two observation imply two rules about any set of FDs that imply a given FD. First, one of the FDs in the set must have the same attribute to the right of the arrow as the FD being implied. Secondly, one of the FDs must have at least one of the attributes to the left of the arrow in the implied FD, and no attributes that are not on the left side of the implied FD.

[1] Since any FD only has the attribute to the right of the arrow with a 0, any FD that rules out location 2 (A, ~B, C, D, E) must have B to the right of the arrow. FIX!



	A	B	C	D	E
0	1	1	1	1	1
1	0	1	1	1	1
2	1	0	1	1	1
3	0	0	1	1	1
4	1	1	0	1	1
5	0	1	0	1	1
6	1	0	0	1	1
7	0	0	0	1	1
8	1	1	1	0	1
9	0	1	1	0	1
10	1	0	1	0	1
11	0	0	1	0	1
12	1	1	0	0	1
13	0	1	0	0	1
14	1	0	0	0	1
15	0	0	0	0	1
16	1	1	1	1	0
17	0	1	1	1	0
18	1	0	1	1	0
19	0	0	1	1	0
20	1	1	0	1	0
21	0	1	0	1	0
22	1	0	0	1	0
23	0	0	0	1	0
24	1	1	1	0	0
25	0	1	1	0	0
26	1	0	1	0	0
27	0	0	1	0	0
28	1	1	0	0	0
29	0	1	0	0	0
30	1	0	0	0	0
31	0	0	0	0	0

Exercises

Exercise 1 Referring to the attribute space representation to the left, which locations in the attribute space are disallowed by the FD,

A, B → D?

Exercise 2 Which locations are disallowd by the FD,

B → C?

Exercise 3 Which locations are disallowd by the FD,

B, C, E → A?

Exercise 4 Is there a relationship between the number of attributes in the FD and the number of locations disallowed? Is the number of attributes in the FD proportional to the number of locations disallowed?

Exercise 5 Give a functional dependency that would forbid row 16.

Exercise 6 Give a functional dependency that would forbid row 24.

Exercise 7 Give a functional dependency that would forbid row 8 and row 12.

Exercise 8 Give two different functional dependencies that would forbid row 10.

A Compact Representation for FDs

A=1

	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

B=1

	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

C=1

	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

A=0

	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

C=1 & A=0

	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

To the left are illustrations for a simple attribute space of only three attributes. The first table illustrates locations with A=1 (colored red), and B=1 (colored yellow).

The locations disallowed by the FD,

$$C \rightarrow A$$

are those with C=1 and A=0 (colored green).

The set of rows with C=1 is {4, 5, 6, 7}.

The set of rows with A=0 is {0, 2, 4, 6}.

The set of rows with C=1 and A=0 is {4, 6}

Note that the set of rows for C=1 and A=0 is equal to the intersection of the set of rows for C=1 and the set of rows for A=0. That is,

$$\{4, 6\} = \{4, 5, 6, 7\} \cap \{0, 2, 4, 6\}.$$

The locations set for a combination of attributes can be found by taking the intersection of the locations set for the individual attributes.

We can find the set of locations disallowed by an FD by taking the intersection of the individual attributes for the FD.

Example

The set of locations disallowed by the FD,

$$B, C \rightarrow A$$

is the intersection of the locations sets for B, C, and ~A.

	A	B	C	D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

The table at left represents an attribute space for the four attributes A, B, C, D.

The FD,

B, C → D

disallows locations with the attribute combination,

B and C and ~D

As we have noted above, the set of locations for (B and C and ~D) is the intersection of the sets of locations for B, and C, and ~D.

The set of locations for B is all ofthe locations where B=1 (colored yellow).

The set of locations for C is all of the locations where C=1 (gray shading).

The set of locations for ~D is all of the locations where D=0 (colored red).

The intersection of those three sets is the set of locations where B=1, C=1, D=0, which are the locations colored yellow, gray shaded, and colored red (locations 6 and 7).

	A	B	C	D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

We have seen that an attribute (as well as its negation) is represented in an attribute space by a set of locations.

We have seen that the combination of multiple attributes is represented in an attribute space by set of locations corresponding to the intersection of the sets of locations for individual attributes.

An attribute space for k attributes has 2^k locations. So each set of locations is a set of integers in the range 0 to 2^k-1. Therefore the set of locations for an attribute (or its negation) can be represented by a bitmap of 2^k bits. Furthermore, the intersection of two sets of locations corresponds to theAND'd bitmaps for the two sets of locations.

For example, the set of locations for,

A and C and ~D

can be represented by the bitmaps for A, C, and ~D, all AND'd together.

Since such bitmap manipulations are very easy to do with a computer program, bitmaps are a very straight-forward and simple method for representing functional dependencies.

The starting point is defining bitmaps for single attributes. Notice in the table at left that the locations for A are alternating locations starting at 1 (colored yellow), the locations for B are two 1's in a row starting at 2, and alternating with two 0's in a row.

If α is the i^{th} attribute (in the range 0 to k-1 for k attributes), and $\alpha[j]$ is the value of α at the j^{th} location,

$$\alpha[j] = (j \bmod step) /(step/2))$$

where step = 2^{i+1}, with rounding down on division.

For example, to calculate B[9] in the table at left,

step = 2^{1+1} (B is attribute number 1 in the range 0..3)

and B[9] = (9 mod 4)/(4/2) = 1/2) = 0.

To calculate C[7], step = 2^{2+1} (C is attribute number 2),

and C[7] = (7 mod step)/(step/2) = (7 mod 8)/(8/2) = 7/4 = 1

Note also that the calculation for a negative attribute, such as ~B[j], is B[j] $+_2$ 1, or (B[j] + 1) mod 2. So if B[j] = 0, ~B[j] = 1, and if B[j] = 1, ~B[j] = 0.

Hence we have,

$$\sim\alpha[j] = (j \bmod step) /(step/2)) +_2 1$$

The formulas above define the bitmap for each attribute, and for the negation of each attribute, in the attribute space. Once the bitmap for each attribute is defined we can easily calculate the bitmap for any FD.

In general, for an FD,

$$\alpha_i \text{ and ... and } \alpha_j \rightarrow \beta$$

equals

$$\text{not } (\alpha_i \text{ and ... and } \alpha_j \text{ and } {\sim}\beta)$$

the bitmap for the FD is inverse[bitmap(α_i) ∩ ... ∩ bitmap(α_j) ∩ bitmap(~β)].

We find it convenient to ignore the outermost inverse() operation and consider,

$$\text{bitmap}(\alpha_i) \cap ... \cap \text{bitmap}(\alpha_j) \cap \text{bitmap}({\sim}\beta)$$

as the bitmap of the locations that the FD *disallows*, which we consider the essential meaning of the FD.

Once we have a method for calculating the bitmaps for FDs, we can easily determine minimum covers for a set of FDs.

It is also simple to determine, for any given set of FDs, and a single FD in the set, which subsets of the remaining FDs in the set imply the single FD.

f^+

Solutions to Exercises

1. Rows with A=1, B=1, and D=0, which are rows: 8, 12, 24, 28
2. Rows with B=1 and C=0, which are rows: 4, 5, 12, 13, 20, 21, 28, 29
3. Rows with B=1, C=1, E=1, A=0, which are rows: 1, 9
4. Inversely proportional
5. Any FD with E to the right of the arrow, such as A →E, A, B →E, C, D →E etc.
6. Any FD with D or E to the right of the arrow, and any combination of A, B, C to the left.
7. Any FD with D to the right of the arrow with any combination of A, B, C to the left.
8. Any two FDs with B or D to the right of the arrow, and any combination of A, B, E to the left.