

Dependencies

Remy Wang

April 2026

A key difference between a DB and a spreadsheet is that, in a DB, data is stored in multiple, interconnected tables. This raises the question of how to organize data across tables.

Let's consider our favorite pets again:

ID	name	age	food
1	Casa	9	chicken
1	Casa	9	grass
2	Kira	7	fish
2	Kira	7	grass

Here we have a new column storing their favorite food. But because each of them likes more than one kinds of food, we need two rows per cat, duplicating their name and age.

You may be tempted to replace the duplicated data with NULLs to save space:

ID	name	age	food
1	Casa	9	chicken
1	NULL	NULL	grass
2	Kira	7	fish
2	NULL	NULL	grass

But now the natural query looking for cats who like grass breaks!

```
SELECT name
FROM pets
WHERE food = 'grass'
```

So let's go back to duplicating the entries.

ID	name	age	food
1	Casa	9	chicken
1	Casa	9	grass
2	Kira	7	fish
2	Kira	7	grass

Can you think of a way to break up the data into multiple tables to remove the redundancy?

We can have two tables, one storing name and age, and another storing food:

ID	name	age
1	Casa	9
2	Kira	7

ID	food
1	chicken
1	grass
2	fish
2	grass

Now *name* and *age* are no longer repeated! You may notice the duplicated IDs in the second table, but that's unavoidable, and an integer takes up little space.¹

¹There's also a duplicate of *grass* which can be eliminated. Try to do that after you learn about decomposition.

Taking a step back, we see the root cause of the data redundancy is that although ID *uniquely determines* name and age, we repeat the information for every occurrence of ID:

ID	name	age	food
1	Casa	9	chicken
1	Casa	9	grass
2	Kira	7	fish
2	Kira	7	grass

Formally, there is a **functional dependency** (FD) $ID \rightarrow name$, as well as $ID \rightarrow age$.

In general, an FD $X \rightarrow Y$ (where X and Y are sets of attributes) holds for a table t , if the value of X uniquely determines the value of Y .

In our example, we also have $ID \rightarrow \text{name, age}$, but $ID \rightarrow \text{food}$ does not hold.

For an example with multiple attribute on the left, consider a table storing GPS coordinates and zipcode:

long.	lat.	zip
34N	118W	90095
47N	122W	98195

Neither the longitude nor the latitude alone uniquely determines the zipcode, but they do together, so we have long., lat. \rightarrow zip

There's also some “trivial” FDs:

- ▶ For any attribute x , $x \rightarrow x$ holds
- ▶ For any sets $Y \subseteq X$, $X \rightarrow Y$ holds

We say k is a *key* if $k \rightarrow x$ holds for every attribute x in the table.² In our example, ID is a key after we break the table up, but it was not when we had one big table.

ID	name	age
1	Casa	9
2	Kira	7

ID	food
1	chicken
1	grass
2	fish
2	grass

²rigorously, a key must be *minimal*, otherwise it's a *superkey*.

“Breaking the table up” is formally known as *decomposition* or *normalization*, and the goal is to exploit FDs to remove redundancy.

In our example, we decomposed with the FD $ID \rightarrow \text{name, age}$.

In general we want to decompose so that for every FD $X \rightarrow Y$, X is a key.³⁴

³Or $X \rightarrow Y$ is *trivial*, meaning $Y \subseteq X$.

⁴More rigorously, X is a *superkey* meaning it contains a key.

But we could have decomposed in 2 steps: first with ID \rightarrow name, then with ID \rightarrow age:

ID	name
1	Casa
2	Kira

ID	age
1	9
2	7

ID	food
1	chicken
1	grass
2	fish
2	grass

This however takes up more space because the ID column is now duplicated.

Intuitively, when we break up, we want to take as much as possible, i.e., we want to decompose with an FD that has many attributes on the right.

To do this, we compute the *FD closure*: given a set of FDs Φ , the *closure* X^+ of a set X is the largest set such that $X \rightarrow X^+$ follows from Φ

Here, an FD ϕ *follows from* Φ if ϕ holds in any table t satisfying Φ .

That's all very abstract, so let's look at some cats. Suppose we have a table with these columns:

name	birth day	age	astrology sign	personality
...

Then we have:

$\text{bday} \rightarrow \text{age}$

$\text{bday} \rightarrow \text{astro}$

$\text{astro} \rightarrow \text{personality}$

We can deduce $\text{bday} \rightarrow \text{personality}$ by *composing* the FDs $\text{bday} \rightarrow \text{astro}$ and $\text{astro} \rightarrow \text{personality}$.

In general, we can deduce additional FDs using the “Armstrong axioms”:

- ▶ **Reflexivity:** $X \rightarrow Y$ if $Y \subseteq X$
- ▶ **Augmentation:** if $X \rightarrow Y$, then $XZ \rightarrow YZ$
- ▶ **Transitivity:** if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

We've seen examples of reflexivity and transitivity. Augmentation is also intuitive: from $\text{bday} \rightarrow \text{age}$ we can conclude $\text{bday, name} \rightarrow \text{age, name}$.

Actually, you don't need to remember the rules. There's a even simpler way to directly compute X^+ . The intuition can be explained with the *tableux method*

Given the FDs $\text{bday} \rightarrow \text{age}$, $\text{bday} \rightarrow \text{astro}$, and $\text{astro} \rightarrow \text{pers.}$, let's compute bday^+ . First we write down a table with just two rows:

name	birth day	age	astrology sign	personality
Casa	3/20/2017	9	Pisces	dreamy
?	3/20/2017	?	?	?

Then, we *apply* each FD to see if we can infer the unknown values.

Step 1: Apply bday \rightarrow age. Both rows share the same birthday, so the unknown age must equal 9.

name	birth day	age	astrology sign	personality
Casa	3/20/2017	9	Pisces	dreamy
?	3/20/2017	9	?	?

Step 2: Apply bday → astro. Both rows share the same birthday, so the unknown astrology sign must equal Pisces.

name	birth day	age	astrology sign	personality
Casa	3/20/2017	9	Pisces	dreamy
?	3/20/2017	9	Pisces	?

Step 3: Apply astro → pers. Both rows now share the same astrology sign, so the unknown personality must equal dreamy.

name	birth day	age	astrology sign	personality
Casa	3/20/2017	9	Pisces	dreamy
?	3/20/2017	9	Pisces	dreamy

No more FDs can fill in new values — we've reached a *fixpoint*.

The final result is:

name	birth day	age	astrology sign	personality
Casa	3/20/2017	9	Pisces	dreamy
?	3/20/2017	9	Pisces	dreamy

This tell us the closure $\text{bday}^+ = \{\text{bday}, \text{age}, \text{astro.}, \text{pers.}\}$, i.e., birthday uniquely determines all of these attributes.

More abstractly, the algorithm works like this:

```
def clos(FDs, V):  
    C = V  
    while C changes:  
        for (X, Y) in FDs:  
            if C contains X:  
                add Y to C  
    return C
```

In words: we start with the set V , then keep applying each FD to grow the set, until doing so doesn't change it anymore.

We can use the closure algorithm to check if a given FD $\varphi = X \rightarrow Y$ follows from a given set of FDs Φ , denoted $\Phi \vdash \varphi$:

1. Compute the closure X^+ of X over Φ
2. Check if $Y \subseteq X^+$
3. If so, $\Phi \vdash \varphi$, otherwise $\Phi \not\vdash \varphi$

That's a lot of new concepts! Let's step back and think about our motivation.

Our original table had *redundancy*, which we now understand is caused by *functional dependencies*:

ID	name	age	food
1	Casa	9	chicken
1	Casa	9	grass
2	Kira	7	fish
2	Kira	7	grass

In particular, we have FDs like $ID \rightarrow name$ where the left-hand-side ID is *not* a key.

To fix the redundancy, we break the table up according to the FDs:

ID	name	age
1	Casa	9
2	Kira	7

ID	food
1	chicken
1	grass
2	fish
2	grass

We also want to “take as much as possible” when breaking up, so we want to compute the *closure* which tells us *all* the dependent attributes.

So our algorithm so far looks like this:

- ▶ Find any FD $X \rightarrow Y$ such that $Y \not\subseteq X$ and X does not contain a key
- ▶ Compute the closure X^+ of X
- ▶ Move X^+ to a new table while keeping X in the old one

Let's also clarify some terminology:

- ▶ A single attribute k is a *key* if $k \rightarrow y$ for every other attribute y

Let's also clarify some terminology:

- ▶ A single attribute k is a *key* if $k \rightarrow y$ for every other attribute y
- ▶ We can also have multi-attribute keys (composite keys):

Let's also clarify some terminology:

- ▶ A single attribute k is a *key* if $k \rightarrow y$ for every other attribute y
- ▶ We can also have multi-attribute keys (composite keys):
 - ▶ K is a *superkey*⁵ if $K \rightarrow y$ for every y

⁵superset of a key

Let's also clarify some terminology:

- ▶ A single attribute k is a *key* if $k \rightarrow y$ for every other attribute y
- ▶ We can also have multi-attribute keys (composite keys):
 - ▶ K is a *superkey*⁵ if $K \rightarrow y$ for every y
 - ▶ A superkey K is a *key* if no $K' \subset K$ is a superkey, i.e. K is *minimal*

⁵superset of a key

Let's also clarify some terminology:

- ▶ A single attribute k is a *key* if $k \rightarrow y$ for every other attribute y
- ▶ We can also have multi-attribute keys (composite keys):
 - ▶ K is a *superkey*⁵ if $K \rightarrow y$ for every y
 - ▶ A superkey K is a *key* if no $K' \subset K$ is a superkey, i.e. K is *minimal*

⁵superset of a key

Let's also clarify some terminology:

- ▶ A single attribute k is a *key* if $k \rightarrow y$ for every other attribute y
- ▶ We can also have multi-attribute keys (composite keys):
 - ▶ K is a *superkey*⁵ if $K \rightarrow y$ for every y
 - ▶ A superkey K is a *key* if no $K' \subset K$ is a superkey, i.e. K is *minimal*

The idea is, once we have a composite key K , we can always keep adding stuff to K to get another superkey, but K is more interesting because it doesn't contain "junk".

⁵superset of a key

Let's practice this on the larger table as well:

name	birth day	age	astrology sign	personality
...

1. We have bday \rightarrow age yet bday is not a key

Let's practice this on the larger table as well:

name	birth day	age	astrology sign	personality
...

1. We have $\text{bday} \rightarrow \text{age}$ yet bday is not a key
2. Compute $\text{bday}^+ = \{\text{bday}, \text{age}, \text{astro}, \text{pers.}\}$

Let's practice this on the larger table as well:

name	birth day	age	astrology sign	personality
...

1. We have $\text{bday} \rightarrow \text{age}$ yet bday is not a key
2. Compute $\text{bday}^+ = \{\text{bday}, \text{age}, \text{astro}, \text{pers.}\}$
3. Break the table into two

Let's practice this on the larger table as well:

name	birth day	age	astrology sign	personality
...

1. We have $\text{bday} \rightarrow \text{age}$ yet bday is not a key
2. Compute $\text{bday}^+ = \{\text{bday}, \text{age}, \text{astro}, \text{pers.}\}$
3. Break the table into two

Let's practice this on the larger table as well:

name	birth day	age	astrology sign	personality
...

1. We have $\text{bday} \rightarrow \text{age}$ yet bday is not a key
2. Compute $\text{bday}^+ = \{\text{bday}, \text{age}, \text{astro}, \text{pers.}\}$
3. Break the table into two

name	birth day
...	...

birth day	age	astrology sign	personality
...

But we're not done! In the second table, we still have redundancy:

birth day	age	astrology sign	personality
3/20/2017	9	Pisces	dreamy
2/19/2016	10	Pisces	dreamy

But we're not done! In the second table, we still have redundancy:

birth day	age	astrology sign	personality
3/20/2017	9	Pisces	dreamy
2/19/2016	10	Pisces	dreamy

To fix this, we *repeat the same process* to decompose this table

1. astro \rightarrow pers. yet astro is not a key

But we're not done! In the second table, we still have redundancy:

birth day	age	astrology sign	personality
3/20/2017	9	Pisces	dreamy
2/19/2016	10	Pisces	dreamy

To fix this, we *repeat the same process* to decompose this table

1. astro \rightarrow pers. yet astro is not a key
2. Compute $\text{astro}^+ = \{\text{astro}, \text{pers.}\}$

But we're not done! In the second table, we still have redundancy:

birth day	age	astrology sign	personality
3/20/2017	9	Pisces	dreamy
2/19/2016	10	Pisces	dreamy

To fix this, we *repeat the same process* to decompose this table

1. astro \rightarrow pers. yet astro is not a key
2. Compute $\text{astro}^+ = \{\text{astro}, \text{pers.}\}$
3. *Factor out* $\{\text{astro}, \text{pers.}\}$

We end up with 3 tables that cannot be decomposed further:

name	birth day
...	...

birth day	age	astrology sign
...

astrology sign	personality
...	...

The general decomposition algorithm is also a *fixpoint* algorithm:

```
def decompose(R):  
    S = attributes of R  
    find nontrivial  $X \rightarrow Y$  s.t. X contains no key  
    if not found:  
        return  
    else:  
        C = clos(X)  
        break R into  $R_1(C)$ ,  $R_2(S - (C - X))$   
        decompose(R1)  
        decompose(R2)
```

At the end of this process, the tables are in *Boyce-Codd Normal Form*, meaning that any FD $X \rightarrow Y$ that still hold in any table will satisfy either:

- ▶ $Y \subseteq X$ (“trivial” FD)
- ▶ Some $K \subseteq X$ is a key (X is a *superkey*)