Database normalization algorithm

Yehor KOROTENKO

29th Apr, 2025

created in Curvenote

Okay, let's simplify the database normalization process and walk through an example.

The Simple Goal:

Imagine you have a big spreadsheet (a table) with lots of repeated information. This repetition causes headaches:

- **Wasted space:** Storing the same info over and over.
- **Update problems:** If something changes (like a cinema's phone number), you have to find and fix it *everywhere* it appears, maybe missing some.
- **Deletion problems:** Deleting one piece of information (like the last movie showing at a cinema) might accidentally delete other unrelated info (like the cinema's address).
- **Insertion problems:** You might not be able to add some information (like a new cinema) until you have other information ready (like a movie showing there).

Normalization fixes this by breaking the big table into smaller, related tables where each piece of information is stored only once.

The Simplified Algorithm (Aiming for BCNF - a common goal):

1. **Start:** Look at your initial table and the rules (Functional Dependencies - FDs) that say "If you know this, you know that" (e.g., `StudentID` $\rightarrow StudentName$).

1. **`Find the Key:`** `Figure out the main identifier (Candidate Key) for the` *`current`* `table. This is the minimum set of columns that uniquely identifies any row.`
2. **`Check the Rules (FDs):`** `Look at` *`each`* `FD (rule) for the table: X` $\rightarrow Y$.

    3. **`Is the LEFT side (X) a potential key (a Superkey) for this table?`** `(A superkey is a candidate key or a candidate key plus extra columns).`

If ALL Rules Pass: If *every* FD in the table has a left side (X) that is a superkey, the table is good (in BCNF). Stop for this table.

If ANY Rule Fails (Decompose!):

- Pick *one* rule `X` $\rightarrow YwhereXis\texttt{NOT}asuperkey$.
- **`Split the table:`**
  - `Create a` **`new table`** `with only the columns from the rule: (X, Y). X usually becomes the key of this new table.`
  - **`Remove`** `the right side (Y) from the` **`original table`**`, but` *`keep`* `the left side (X) in it.`

- **Repeat:** Now you have two (or more) smaller tables. Go back to Step 2 for *each* of these smaller tables and check them individually. Keep splitting until all tables pass the check in Step 4.

Concrete Example (Using the `BD-Cine` Idea from the Slides):

Let's say we start with one big table to track which actors are in which movies shown at which cinemas, including director and cinema details.

Step 0: Start

- **Initial Table:** `Showings(Titre, M-e-S, Acteur, Nom-Ciné, Adr, Tél)`
  - `Titre`: Movie Title
  - `M-e-S`: Director (Metteur en scène)
  - `Acteur`: Actor Name
  - `Nom-Ciné`: Cinema Name
  - `Adr`: Cinema Address
  - `Tél`: Cinema Telephone
- **Rules (FDs):**
  - FD1: `Titre` $\rightarrow M-e-S(Amovietitledeterminesitsdirector)$
  - FD2: `Nom-Ciné` $\rightarrow Adr, Tl(Acinemanamedeterminesitsaddressandphone)$

Step 1: Find the Key

- To uniquely identify one specific row (a specific actor in a specific movie at a specific cinema), we need a combination of columns. Based on the attributes, a likely candidate key is `{Titre, Acteur, Nom-Ciné}`. Knowing these three tells you exactly which row you're talking about.

Step 2 & 3: Check the Rules (FDs) against `Showings`

- **Check FD1:** `Titre` $\rightarrow M-e-S$
  - Is the left side (`Titre`) a superkey of `Showings`? No. Knowing just the Title doesn't uniquely identify a row (many actors could be in the same movie, shown at many cinemas).
  - **VIOLATION!**

Check FD2: `Nom-Ciné` $\rightarrow Adr, Tl$

```
Is the left side (Nom-Ciné) a superkey of Showings? No. Knowing just the Cinema
Name doesn't uniquely identify a row (it could show many movies with many actors).
VIOLATION!
```

Step 4: If ALL Rules Pass: They don't pass.

Step 5: Decompose (Pick FD1)

- Violating Rule: `Titre` $\rightarrow M-e-S$
- `Split:`
  - `New Table 1: Movies(Titre, M-e-S)`
    - `Attributes = {Titre, M-e-S}`

    * Rule inside: `Titre` $\rightarrow M - e - S$
  - **Modify Original Table:** Remove M-e-S. Let's call it Showings_Temp.
    * Showings_Temp(Titre, Acteur, Nom-Ciné, Adr, Tél)
    * Rules remaining that apply: Nom-Ciné $\rightarrow Adr, Tl$

Repeat the Process for the New Tables (`Movies` and `Showings_Temp`)

- **Analyze `Movies(Titre, M-e-S)`:**
  - Step 1 (Key): `{Titre}` is the key.
  - Step 2/3 (Check Rules): Rule is `Titre` $\rightarrow M - e - S.Istheleftside($`Titre`$)asuperkeyof$`Movies`$?Yes, it'sthekey.$
  - Step 4 (Pass?): Yes. **Movies table is in BCNF.**
- **Analyze `Showings_Temp(Titre, Acteur, Nom-Ciné, Adr, Tél)`:**
  - Step 1 (Key): The key is still `{Titre, Acteur, Nom-Ciné}` (we need all three to identify a specific actor in a specific movie at a specific cinema).
  - Step 2/3 (Check Rules): Rule is `Nom-Ciné` $\rightarrow Adr, Tl.Istheleftside($`Nom-Ciné`$)asuperkeyof$`Showings_Temp`$?No.$
  - VIOLATION!
  - Step 4 (Pass?): No.
  - Step 5 (Decompose using Nom-Ciné $\rightarrow Adr, Tl$) :
    - **Split:**
      * **New Table 2:** Cinemas(Nom-Ciné, Adr, Tél)
        · Attributes = {Nom-Ciné, Adr, Tél}
        · Rule inside: Nom-Ciné $\rightarrow Adr, Tl$
      * **Modify Showings_Temp Table:** Remove Adr, Tél. Let's call it Performances.
        · Performances(Titre, Acteur, Nom-Ciné)
        · Rules remaining: None from the original set apply *within* just these attributes.

Repeat the Process for New Tables (`Cinemas` and `Performances`)

- **Analyze `Cinemas(Nom-Ciné, Adr, Tél)`:**
  - Step 1 (Key): `{Nom-Ciné}` is the key.
  - Step 2/3 (Check Rules): Rule is `Nom-Ciné` $\rightarrow Adr, Tl.Istheleftside($`Nom-Ciné`$)asuperkeyof$`Cinemas`$?Yes, it'sthek$
  - Step 4 (Pass?): Yes. **Cinemas table is in BCNF.**
- **Analyze `Performances(Titre, Acteur, Nom-Ciné)`:**
  - Step 1 (Key): `{Titre, Acteur, Nom-Ciné}` is the key.
  - Step 2/3 (Check Rules): No remaining (non-trivial) FDs to check from our original list.
  - Step 4 (Pass?): Yes (no rules to violate). **`Performances` table is in BCNF.**

Final Result:

We decomposed the original big table `Showings` into three smaller tables, all in BCNF:

1. `Movies(Titre, M-e-S)` (Key: `Titre`)

- Stores movie titles and their directors. Director info is stored only once per movie.

2. `Cinemas(Nom-Ciné, Adr, Tél)` (Key: `Nom-Ciné`)
   - Stores cinema names and their details. Address/phone are stored only once per cinema.
3. `Performances(Titre, Acteur, Nom-Ciné)` (Key: `{Titre, Acteur, Nom-Ciné}`)
   - Stores the relationship: which actor was in which movie shown at which cinema. This table links the other two tables together using `Titre` and `Nom-Ciné` (which act as foreign keys here).

Now, redundancy is reduced, and update anomalies are avoided!