Perfect 👍 — let’s take **one realistic example** and walk it **step-by-step from Unnormalized → 1NF → 2NF → 3NF → BCNF → 4NF → 5NF**, including practical scenarios for **all** normal forms.

**🏢 Example: “Employee Project Management System”**

**🎯 Scenario**

A company stores information about **employees**, the **projects** they work on, and the **skills** they possess.

We’ll start from an **unnormalized table** and gradually normalize it through all forms.

**🔹 UNNORMALIZED FORM (UNF)**

| **EmpID** | **EmpName** | **Projects** | **Skills** | **Dept** |
| --- | --- | --- | --- | --- |
| E1 | Ramesh | P1, P2 | Java, SQL | IT |
| E2 | Suresh | P2 | Python | IT |
| E3 | Priya | P3 | HTML, CSS | Design |

**🔸 Problems:**

* Multi-valued attributes (Projects, Skills)
* Data redundancy
* Difficult to search or update

**🔹 1NF (First Normal Form)**

**Rule:**  
➡️ Each cell must have atomic (single) values.  
➡️ Each row must be unique.

**Fix:** Split multiple values into separate rows.

| **EmpID** | **EmpName** | **Project** | **Skill** | **Dept** |
| --- | --- | --- | --- | --- |
| E1 | Ramesh | P1 | Java | IT |
| E1 | Ramesh | P1 | SQL | IT |
| E1 | Ramesh | P2 | Java | IT |
| E1 | Ramesh | P2 | SQL | IT |
| E2 | Suresh | P2 | Python | IT |
| E3 | Priya | P3 | HTML | Design |
| E3 | Priya | P3 | CSS | Design |

✅ Now every attribute has atomic values.  
**→ 1NF achieved.**

**🔹 2NF (Second Normal Form)**

**Rule:**  
➡️ Table must be in 1NF.  
➡️ No **partial dependency** (non-key column depending on part of a composite key).

**Current key:** (EmpID, Project, Skill)  
→ Dept depends only on EmpID (not full key).

**Fix:** Split table into two.

**1️⃣ Employee Table**

| **EmpID** | **EmpName** | **Dept** |
| --- | --- | --- |
| E1 | Ramesh | IT |
| E2 | Suresh | IT |
| E3 | Priya | Design |

**2️⃣ EmployeeProjectSkill Table**

| **EmpID** | **Project** | **Skill** |
| --- | --- | --- |
| E1 | P1 | Java |
| E1 | P1 | SQL |
| E1 | P2 | Java |
| E1 | P2 | SQL |
| E2 | P2 | Python |
| E3 | P3 | HTML |
| E3 | P3 | CSS |

✅ Every non-key attribute depends on the full primary key → **2NF achieved.**

**🔹 3NF (Third Normal Form)**

**Rule:**  
➡️ Must be in 2NF.  
➡️ No **transitive dependency** (non-key → non-key).

Let’s assume:

* Dept determines **Manager** (i.e., Dept → Manager).

**Fix:** Create a new table for Department info.

**1️⃣ Employee**

| **EmpID** | **EmpName** | **Dept** |
| --- | --- | --- |
| E1 | Ramesh | IT |
| E2 | Suresh | IT |
| E3 | Priya | Design |

**2️⃣ Department**

| **Dept** | **Manager** |
| --- | --- |
| IT | Mr. Rao |
| Design | Ms. Latha |

**3️⃣ EmployeeProjectSkill**

| **EmpID** | **Project** | **Skill** |
| --- | --- | --- |
| E1 | P1 | Java |
| E1 | P1 | SQL |
| E1 | P2 | Java |
| E1 | P2 | SQL |
| E2 | P2 | Python |
| E3 | P3 | HTML |
| E3 | P3 | CSS |

✅ Now, there’s no non-key → non-key dependency.  
**→ 3NF achieved.**

**🔹 BCNF (Boyce–Codd Normal Form)**

**Rule:**  
➡️ For every functional dependency (X → Y), X must be a **superkey**.  
➡️ Stronger than 3NF.

Let’s say:

* Each **Project** has **one Manager**.  
  So, Project → Manager

But Manager is also determined by Dept (Dept → Manager).

Potential conflict:  
If Project is in IT and Design both, multiple Managers can exist.

**Fix:** Create a separate Project table.

**4️⃣ Project Table**

| **Project** | **Manager** |
| --- | --- |
| P1 | Mr. Rao |
| P2 | Mr. Rao |
| P3 | Ms. Latha |

✅ Now each determinant (Project, Dept, EmpID) is a superkey → **BCNF achieved.**

**🔹 4NF (Fourth Normal Form)**

**Rule:**  
➡️ Must be in BCNF.  
➡️ No **multi-valued dependencies** (MVDs) — one entity having independent multi-valued facts.

**Problem:**  
An employee can have multiple **Projects** and multiple **Skills** independently.  
(E.g., Ramesh can work on P1, P2 and also have Java, SQL skills not tied directly.)

So, we must split **Projects** and **Skills**.

**Fix:** Separate independent multi-valued sets.

**5️⃣ EmployeeProject**

| **EmpID** | **Project** |
| --- | --- |
| E1 | P1 |
| E1 | P2 |
| E2 | P2 |
| E3 | P3 |

**6️⃣ EmployeeSkill**

| **EmpID** | **Skill** |
| --- | --- |
| E1 | Java |
| E1 | SQL |
| E2 | Python |
| E3 | HTML |
| E3 | CSS |

✅ Now there’s no independent multi-valued dependency.  
**→ 4NF achieved.**

**🔹 5NF (Fifth Normal Form)**

**Rule:**  
➡️ Must be in 4NF.  
➡️ No **join dependency** — data should not be reconstructible only by joining multiple smaller tables unnecessarily.

**Problem (Join Dependency Example):**  
If an employee can **work on multiple projects**, and each project can **require multiple skills**, we might need to know **which employee uses which skill in which project**.

But this can only be known by joining the three: EmployeeProject, EmployeeSkill, and ProjectSkill.

If we store everything properly in one relation:

**7️⃣ EmployeeProjectSkill (Final Combined Table if needed)**

| **EmpID** | **Project** | **Skill** |
| --- | --- | --- |
| E1 | P1 | Java |
| E1 | P1 | SQL |
| E1 | P2 | Java |
| E1 | P2 | SQL |
| E2 | P2 | Python |
| E3 | P3 | HTML |
| E3 | P3 | CSS |

Now, if every combination can be derived from valid joins (no redundant or missing info), **5NF** is achieved.

✅ **→ 5NF ensures all decompositions are lossless and minimal.**

**🧠 Summary Table**

| **Normal Form** | **Rule** | **Example Problem** | **Solution** |
| --- | --- | --- | --- |
| 1NF | Atomic values | Multi-valued Projects, Skills | Split rows |
| 2NF | No partial dependency | Dept depends only on EmpID | Split Employee info |
| 3NF | No transitive dependency | Dept → Manager | Create Dept table |
| BCNF | Every determinant is superkey | Project → Manager | Create Project table |
| 4NF | No multi-valued dependency | Employee has independent Projects & Skills | Split into EmployeeProject, EmployeeSkill |
| 5NF | No join dependency | Employee–Project–Skill relationships | Use proper join tables |

✅ **Final Normalized Database Schema (up to 5NF)**

1. **Employee(EmpID, EmpName, Dept)**
2. **Department(Dept, Manager)**
3. **Project(Project, Manager)**
4. **EmployeeProject(EmpID, Project)**
5. **EmployeeSkill(EmpID, Skill)**