**Documentation on Database Normalization in PostgreSQL**

**1. What is Database Normalization?**

Database Normalization in PostgreSQL is a **step-by-step process** of organizing tables to reduce redundancy, remove anomalies, and ensure data integrity.  
It is applied in levels called **Normal Forms (1NF → 5NF, BCNF)**.

**Why Normalization is Needed?**

Without normalization, a database may contain problems such as:

* **Data redundancy:** The same data stored in multiple places.
* **Update anomalies:** Changes need to be made in several places.
* **Insert anomalies:** New data cannot be added without unnecessary or missing data.
* **Delete anomalies:** Removing one piece of data may result in losing valuable related data**.**

**Step 1: Unnormalized Form (UNF)**

In UNF, data contains **repeated groups** and **non-atomic values**.

**Table: Student Records (UNF)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Student\_ID** | **Name** | **Courses** | **Hobbies** | **Teacher** | **Dept** |
| 1 | Rashi | Math, Science | Reading, Sports | Dr. A | Science |
| 2 | Rohit | English, History | Music | Dr. B | Arts |

**Problems:**

* Multiple values in single columns (Courses, Hobbies).
* Data redundancy if a student takes more courses.

**Step 2: First Normal Form (1NF)**

* **Rule:** Data must be **atomic** (no multi-valued columns).
* **Converted Table (1NF)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **StudentID** | **Name** | **Course** | **Hobby** | **Teacher** | **Dept** |
| 1 | Rashi | Math | Reading | Dr. A | Science |
| 1 | Rashi | Math | Sports | Dr. A | Science |
| 1 | Rashi | Science | Reading | Dr. A | Science |
| 1 | Rashi | Science | Sports | Dr. A | Science |
| 2 | Rohit | English | Music | Dr. B | Arts |
| 2 | Rohit | History | Music | Dr. B | Arts |

* **Fix:** Atomic values but still redundant.

**Step 3: Second Normal Form (2NF)**

**Rule:** Must be in 1NF + no **partial dependency** (non-key attribute depending only on part of a composite key).

**Fix:** Separate into different tables.

**Students Table**

|  |  |  |
| --- | --- | --- |
| **Student\_ID** | **Name** | **Dept** |
| 1 | Rashi | Science |
| 2 | Rohit | Arts |

|  |  |  |
| --- | --- | --- |
| **CourseID** | **Course** | **Teacher** |
| C1 | Math | Dr. A |
| C2 | Science | Dr. A |
| C3 | English | Dr. B |
| C4 | History | Dr. B |

**Step 4: Third Normal Form (3NF)**

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

**Problem:** In Students table, Dept depends on Student, but Dept has its own details.

**Fix:** Create separate Department table.

**Departments Table**

|  |  |
| --- | --- |
| **DeptID** | **DeptName** |
| D1 | Science |
| D2 | Arts |

**Updated Students Table**

|  |  |  |
| --- | --- | --- |
| **StudentID** | **Name** | **DeptID** |
| 1 | Rashi | D1 |
| 2 | Rohit | D2 |

**Step 5: Boyce-Codd Normal Form (BCNF)**

**Rule:** Every determinant must be a **candidate key**.

**Problem:** In Courses table, Teacher determines Course, but Teacher may teach multiple courses.

**Fix:** Separate Teachers.

**Teachers Table**

|  |  |
| --- | --- |
| **TeacherID** | **Teacher** |
| T1 | Dr. A |
| T2 | Dr. B |

**Updated Courses Table**

|  |  |  |
| --- | --- | --- |
| **CourseID** | **Course** | **TeacherID** |
| C1 | Math | T1 |
| C2 | Science | T1 |
| C3 | English | T2 |
| C4 | History | T2 |

**Step 6: Fourth Normal Form (4NF)**

**Rule:** No **multi-valued dependencies**.

**Problem:** A student can have multiple hobbies and multiple courses (independent facts).

**Fix:** Keep Student\_Hobbies and Student\_Courses separate (already done in 2NF).  
Now no table has multiple independent facts.

**Step 7: Fifth Normal Form (5NF)**

**Rule:** No **join dependency** → tables should not be decomposable further without losing meaning.

**Problem Example:** Student–Course–Project

❌ Single Table:

|  |  |  |
| --- | --- | --- |
| **Student** | **Course** | **Project** |
| Rashi | Math | AI |
| Rashi | Science | Robotics |
| Rohit | English | NLP |

**Issue:**  
This table mixes **three independent relationships** → Student–Course, Course–Project, Student–Project.  
If one row is updated incorrectly, it can cause redundancy or inconsistency.

**Comparison of Normal Forms**

|  |  |  |  |
| --- | --- | --- | --- |
| **Normal Form** | **Rule** | **Problem Solved** | **Example Fix** |
| **1NF** | Atomic values | Removes repeating groups | Split multi-value fields |
| **2NF** | No partial dependency | Reduces redundancy | Student–Course separation |
| **3NF** | No transitive dependency | Removes non-key dependency | Separate Dept table |
| **BCNF** | Every determinant is key | Stronger 3NF | Teacher separated |
| **4NF** | No multi-valued dependency | Avoids multiple independent facts | Split Courses & Hobbies |
| **5NF** | No join dependency | Prevents false data recombination | Student–Course–Project separation |

**Best Practices in PostgreSQL**

* Normalize up to **3NF/BCNF** for most real-world applications.
* Use **4NF/5NF** in advanced scenarios (projects, logistics, research).
* Always use **Primary Keys, Foreign Keys** to maintain integrity.
* Balance **normalization vs performance** (sometimes denormalization is faster).

**Final Summary**

* **Normalization** ensures a clean and efficient database structure.
* It reduces redundancy and avoids anomalies.
* PostgreSQL supports all forms (1NF → 5NF).
* **Trade-off:** Higher normalization = less redundancy but more JOINs.
* **Practical Approach:** Aim for **3NF/BCNF**, go beyond if data is highly complex.