**Enhancing Library Database Design through Normalization**

**Mahmoud Hussein Shee  
C110156202  
University of the People**[CS 2203-01 - AY2026-T1](https://my.uopeople.edu/course/view.php?id=9202#section-2)[Stella Chibuike-Ezike (Instructor)](https://my.uopeople.edu/user/view.php?id=376729&course=9202) **Date: Sep 18, 2025**

**Definition of Normalization and Its Importance**

Normalization is when you structure a relational database in order to reduce redundancy and minimize update anomalies along with maintaining data integrity as you organize data into well-defined relations based on functional dependencies (Vidhya, Jeyaram, & Ishwarya, 2016). Normalization makes sure that the database can remain efficient and also consistent. Also, it simplifies database maintenance when redundant or repeated data is eliminated. Scalability is also supported, query accuracy is improved, and logical clarity of the schema is improved (Peterson, 2023b). Databases lacking normalization are prone to inconsistent updates, insertion failures, also to data loss when deleting data.

**Normalization of the Books Relation**

The unnormalized relation is:

**Books (Book\_ID, Title, Author, Genre, Publisher, Publication\_Year, ISBN, Price)**

**1NF (First Normal Form):**  
A relation is in 1NF if it has atomic values and no repeating groups. Since each attribute here is atomic (e.g., one Title per Book\_ID), the table already satisfies 1NF.

**1NF Relation:**  
Books(Book\_ID, Title, Author, Genre, Publisher, Publication\_Year, ISBN, Price)

**2NF (Second Normal Form):**  
A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on the primary key. Assuming **Book\_ID** is the primary key, all attributes depend on Book\_ID. However, if a book can have multiple authors, partial dependency arises. To resolve this, create a separate **BookAuthors** relation.

**2NF Relations:**

* Books(Book\_ID, Title, Genre, Publisher, Publication\_Year, ISBN, Price)
* BookAuthors(Book\_ID, Author)

**3NF (Third Normal Form):**  
A relation is in 3NF if it is in 2NF and there are no transitive dependencies. In this case, **Publisher → PublisherAddress** (if addresses were tracked) would create transitivity. To resolve this, we separate publishers.

**3NF Relations:**

* Books(Book\_ID, Title, Genre, PublisherID, Publication\_Year, ISBN, Price)
* BookAuthors(Book\_ID, Author)
* Publishers(PublisherID, PublisherName)

**BCNF (Boyce-Codd Normal Form):**  
A relation is in BCNF if every determinant is a candidate key. In 3NF, if ISBN uniquely determines a book, then **ISBN → Book\_ID** is a dependency. To eliminate redundancy, create a separate ISBN relation.

**BCNF Relations:**

* Books(Book\_ID, Title, Genre, PublisherID, Publication\_Year, Price)
* BookAuthors(Book\_ID, Author)
* Publishers(PublisherID, PublisherName)
* BookIdentifiers(Book\_ID, ISBN)

**Functional Dependencies and Their Resolution**

The original schema contains the following functional dependencies (FDs):

* Book\_ID → Title, Author, Genre, Publisher, Publication\_Year, ISBN, Price
* ISBN → Book\_ID, Title, Genre, Publisher, Publication\_Year, Price
* Publisher → PublisherName (and possibly other attributes if tracked)

In **1NF**, redundancies exist since ISBN or Publisher could determine other attributes.  
In **2NF**, splitting **BookAuthors** addresses partial dependency between Book\_ID and Author.  
In **3NF**, separating **Publishers** removes transitive dependency where Publisher determines PublisherName.  
In **BCNF**, splitting **BookIdentifiers** ensures ISBN, a determinant, is treated as a candidate key, eliminating redundancy and anomaly risk (Peterson, 2023a).

**Advantages and Drawbacks of Higher Normal Forms**

Achieving higher normal forms like 3NF along with BCNF ensures there is minimal redundancy, strong referential integrity, also clearer data representation (Vidhya et al., 2016). Queries yield consistent results updates happen reliably and analytical scalability is supported by the database. However, drawbacks include more tables possibly reducing performance for read-heavy applications by increasing the number of joins needed for queries (Sengupta & Kumar, 2020). Designers normalize both for integrity and balance of needs. In practice, they control denormalization for efficiency.

**Conclusion**

Normalization transforms a single unstructured relation to become a set of well-defined relations that preserve data integrity, minimize redundancy as well as efficiently manage data. The library’s book data becomes more reliable, consistent, and strong through progressive refinement from 1NF to BCNF. Even though the performance trade-offs may still be possible, higher normal forms do allow for accurate database systems. Database systems which are scalable also do become possible when forms are higher.

**References**

Peterson, R. (2023a, December 9). *Functional dependency in DBMS: What is, types and examples*. Guru99. <https://www.guru99.com/dbms-functional-dependency.html>

Peterson, R. (2023b, December 26). *What is normalization in DBMS (SQL)? 1NF, 2NF, 3NF example*. Guru99. <https://www.guru99.com/database-normalization.html>

Sengupta, S., & Kumar, A. (2020). *Database normalization and performance: A balanced approach.* Journal of Computer Applications, 42(2), 55–62.

Vidhya, V., Jeyaram, G., & Ishwarya, K. (2016). *Database management systems*. Alpha Science International.