Normalization Example: A University Library

The library in this example is a university library with multiple types of resources: books, reference books, multimedia items, and other resources.

- All items must be recorded with the type.
- Multiple copies of the same item can exist. Each item is uniquely identified by its ID prepared by the international library notation.
- Items in the library can be loaned to students and employees.
- Each item type has its standard loan duration. For books it is 2 weeks, for reference books it is 2 days, for multimedia items it is 1 day, and for other resources it has to be defined when the item is first recorded. But the default for the other resources is also 1 day. So we can assume if an item of this type exists it has a 1 day loan duration.
- Users are either students and employees, so their IDs and their user types must be recorded.
- Students can be assigned maximum of 5 items on loan at any time, but employees can be assigned a maximum of 10 items.

Unnormalized Table (UNF)

This table is prepared from the perspective of loans.

LoanID	UserID	First Name	Last Name	User Type	Iteml D	Item Type	Title	Author/ Creator	CopyNum	Loan Duration	Loan Date	Return Date	MaxLoans
L001	S001	Ayşe	Yılmaz	Student	B001	Book	"The Night Circus"	Erin Morgenstern	1	14	2024-10-01	2024-10-15	5
L002	S001	Ayşe	Yılmaz	Student	B002	Book	"The Silent Patient"	Alex Michaelides	1	14	2024-10-01	2024-10-15	5
L003	S001	Ayşe	Yılmaz	Student	T001	Textbook	"Principles of Economics"	N. Gregory Mankiw	1	14	2024-10-02	2024-10-16	5
L004	S002	Mehmet	Kaya	Student	B003	Book	"Where the Crawdads Sing"	Delia Owens	1	14	2024-10-03	2024-10-17	5
L005	S002	Mehmet	Kaya	Student	B004	Book	"The Testaments"	Margaret Atwood	1	14	2024-10-03	2024-10-17	5
L006	S002	Mehmet	Kaya	Student	T002	Textbook	"Economics"	Paul Krugman	1	14	2024-10-04	2024-10-18	5
L007	E001	Ahmet	Demir	Employee	B005	Book	"Educated"	Tara Westover	1	14	2024-10-05	2024-10-19	10
L008	E001	Ahmet	Demir	Employee	B006	Book	"Becoming"	Michelle Obama	1	14	2024-10-05	2024-10-19	10
L009	E001	Ahmet	Demir	Employee	D001	Multimedia	"The Godfather"	Francis Ford Coppola	1	1	2024-10-05	2024-10-06	10
L010	E001	Ahmet	Demir	Employee	T001	Textbook	"Principles of Economics"	N. Gregory Mankiw	2	14	2024-10-06	2024-10-20	10
L011	E001	Ahmet	Demir	Employee	T002	Textbook	"Economics"	Paul Krugman	2	14	2024-10-06	2024-10-20	10
L012	E001	Ahmet	Demir	Employee	T003	Textbook	"Macroeconomics"	Olivier Blanchard	1	14	2024-10-06	2024-10-20	10
L013	E002	Kerem	Aktürkoğlu	Employee	O001	Other	"The Elder Wand"	-	1	1	2024-10-01	2024-10-02	10
L014	E002	Kerem	Aktürkoğlu	Employee	O002	Other	"The Invisibility Cloak"	-	1	1	2024-10-01	2024-10-02	10
L015	E002	Kerem	Aktürkoğlu	Employee	O003	Other	"The Resurrection Stone"	-	1	1	2024-10-01	2024-10-02	10

Preparing the database from the perspective of loans, as shown in the UNF table, can lead to several difficulties in library operations due to data redundancy, inefficiency, and inconsistency. Let's outline the challenges:

1. Data Redundancy and Duplication

- Repetitive Information: Information such as user details (name, ID, user type) and item details (title, author, type) are repeated in multiple rows whenever a loan is made. For example, every time a student borrows a book, their name, ID, and user type are repeated. Similarly, details about the items are duplicated in every loan.
- Wasted Storage: Since the same information is stored repeatedly, this leads to unnecessary storage consumption. As the library grows, this redundancy becomes significant.

2. Data Inconsistency

- Inconsistent Updates: If a student's details or an item's details (e.g., book title, author, loan duration) need to be updated, this would need to be done in every loan record where that student or item appears. Failing to update all records can lead to inconsistent data.
- Error-Prone Data: Repeated fields increase the likelihood of human error during data entry, such as entering the same book title slightly differently across different loans.

3. Inefficient Data Management

- Complex Queries: Operations that are not directly related to loans (e.g., tracking available items, managing users, or performing inventory checks) become cumbersome. Queries need to sift through multiple loan records to extract non-loan-specific information (such as retrieving a list of all items or users).
- Difficulty Managing Copies: Handling multiple copies of the same item (e.g., two copies of the same textbook) becomes difficult. There is no efficient way to track which copies are currently on loan and which are available without manually examining the loan records.
- Limited Flexibility for Non-Loan Operations: Library staff might need to perform operations like adding new items, updating item details, or managing user information independently of the loan system. The current structure intertwines all this data, making it hard to manage resources separately.

4. Loan Restrictions and Item Availability

- Tracking Loan Limits: Since user data is repeated with every loan, enforcing user restrictions (like the maximum number of loans per student or employee) is not straightforward. Determining whether a student has reached their loan limit would require examining all the loan records.
- Item Availability: Similarly, checking the availability of a specific item or resource (e.g., whether all copies of a textbook are on loan) becomes complicated since item data is entangled with the loan data.

5. Maintenance Overhead

- Updating Item Properties: Each item has specific properties, such as the loan duration for different item types. In this structure, if the library updates its policy on loan durations (e.g., extending the loan period for reference books), each loan record would need to be updated manually, making the system inefficient and error-prone.
- Complex Reporting: Generating reports on library inventory, user statistics, or loan trends would be difficult since all the data is stored in a single table with lots of redundant information.

6. Difficulty in Adding New Functionalities

- Scaling Issues: As the library grows, adding new functionalities (such as different types of users, tracking fines, reservations, or renewals) becomes more complicated with a loan-centric design. Every new feature would have to be managed through the same table, increasing complexity.
- Flexibility: A system designed around loans may lack the flexibility to easily accommodate new types of resources or users, different loan policies, or new services such as e-books, which might have different loan requirements.

The UNF structure, designed around loans, makes **library operations difficult and inefficient** due to data duplication, lack of separation between entities (users, items, loans), and difficulty in managing non-loan-related operations like inventory management. To overcome these challenges, the database needs to be **normalized** to separate the concerns of users, items, and loans, making it easier to manage library resources and operations efficiently.

Exercise: Try to conduct some library operations on the UNF table.

Exercise: Transform from UNF to 1NF

To transform the unnormalized table (UNF) into **First Normal Form (1NF)**, we need to follow a few key principles. The goal of 1NF is to eliminate **repeating groups** and ensure that all data fields are **atomic** (indivisible). Here's what we need to do:

Key Changes for 1NF Transformation:

1. **Eliminate Repeating Groups**:
    - In the UNF table, certain fields like CopyNum, LoanDuration, and item-related fields (such as Title, Author/Creator) are repeated multiple times for each loan transaction. Each row represents a combination of user and loan, which causes duplication of information for multiple items under one loan.
    - **Action:** Separate each loaned item into its own row. For example, if a user borrows two books and one multimedia item, these should each be recorded as separate rows instead of combining them in a single row.
2. **Make Fields Atomic**:
    - Ensure that all data is stored in atomic (indivisible) units. In some unnormalized databases, you might see fields like "List of borrowed books" (with multiple items stored in a single field, separated by commas), but in 1NF, each piece of data should be stored separately.
    - **Action:** If any composite fields existed (though none were in this case), they would need to be split into individual atomic fields. For example, each borrowed item should have its own row instead of grouping them together.
3. **Separate Out Multi-Value Attributes**:
    - In UNF, attributes such as ItemType and Title are repeated across multiple rows for different copies of the same item.
    - **Action:** Each loan should only contain the specific details of the item being loaned, without repeating the entire resource information for every loan.
4. **Organize Loan Data Separately**:
    - The information for the **loan transaction** (such as LoanDate, ReturnDate, UserID, etc.) should be linked to each item loaned out. Multiple loans for different items should be tracked as separate rows for the same user, each referencing a specific item.

Note: The **First Normal Form (1NF)** transformation **doesn't inherently require splitting the data into multiple tables**. It focuses on ensuring that each field contains atomic (indivisible) values and that there are no repeating groups within the same table. However, **1NF** still keeps all the data in a **single table**.

What Happens in 1NF:

- **Rows are split** for each loan and item, ensuring that every piece of data is atomic.
- However, there is still **redundancy**, such as repeating user and item information across multiple rows (e.g., a user's name and ID might be repeated for each loan).

Note: The **UNF (Unnormalized Form)** table in this example is actually **already in 1NF**.

Here's why:

1. **No Repeating Groups**:
    - Each loan transaction in the table represents a single item being borrowed, and there are no multiple values or lists of items within a single field. Each item (book, textbook, multimedia item, etc.) has its own row in the table.
2. **Atomic Values**:
    - All fields contain atomic values, meaning that each field holds a single, indivisible piece of data. For example, the ItemType, Title, Author/Creator, LoanDate, etc., are all atomic and contain a single piece of information.

Key Characteristics of the Example that Show It's in 1NF:

- **Each loan is in a separate row**: No multiple items are recorded in the same row.
- **Each field contains atomic values**: There's no field that contains more than one value, such as a list of items, users, or loans in a single column.

The next step would be to **move to 2NF**, which involves eliminating **partial dependencies** by creating separate tables for **users**, **items**, and **loans**.

The Second Normal Form (2NF):

- To transform the database to **Second Normal Form (2NF)**, we need to ensure that the table satisfies the requirements of **1NF** (which it already does) and then eliminate **partial dependencies**. Partial dependencies occur when non-primary key attributes are dependent on part of a composite primary key rather than the whole key.
- In this case, the key steps to achieve **2NF** involve separating data into multiple related tables to remove any dependencies where a non-key attribute depends only on part of a composite key.

Key Changes for 2NF Transformation:

1. **Identify the Composite Key:**
    - In the UNF/1NF table, the composite key is likely the combination of LoanID and ItemID (or LoanID and UserID in certain scenarios), which uniquely identifies each row.
    - Non-key attributes such as UserName, UserType, Title, Author/Creator, etc., do not depend on the entire composite key; instead, they depend on individual parts (e.g., UserName depends on UserID, and Title depends on ItemID).
2. **Eliminate Partial Dependencies:**
    - Attributes that depend only on part of the composite key (e.g., UserName depending on UserID or Title depending on ItemID) should be moved into separate tables.
    - For instance, information about **users** (like their name and type) should be stored in a separate **Users** table, and information about **items** (such as their title, type, and loan duration) should be stored in a separate **Items** table.
    - The **Loans** table will then contain only information directly related to the loan itself (e.g., LoanID, UserID, ItemID, LoanDate, ReturnDate).

Steps to Transform to 2NF:

1. **Create Separate Tables for Users and Items**:
    - Move user-related information (like FirstName, LastName, UserType, MaxLoans) to a separate Users table.
    - Move item-related information (like Title, Author/Creator, ItemType, LoanDuration) to a separate Items table.

2. **Keep a Loan Table** for Loan Transactions:
    - The Loans table will retain only information directly related to each loan, such as LoanID, UserID, ItemID, LoanDate, and ReturnDate.

2NF Table Structure:

1. Users Table:

This table stores information about users (both students and employees), where UserID is the primary key.

UserID	FirstName	LastName	UserType	MaxLoans
S001	Ayşe	Yılmaz	Student	5
S002	Mehmet	Kaya	Student	5
E001	Ahmet	Demir	Employee	10
E002	Kerem	Aktürkoğlu	Employee	10

2. Items Table:

This table stores information about items in the library, where ItemID is the primary key.

ItemID	Title	Author/Creator	ItemType	LoanDuration
B001	"The Night Circus"	Erin Morgenstern	Book	14
B002	"The Silent Patient"	Alex Michaelides	Book	14
T001	"Principles of Economics"	N. Gregory Mankiw	Textbook	14
B003	"Where the Crawdads Sing"	Delia Owens	Book	14
B004	"The Testaments"	Margaret Atwood	Book	14
T002	"Economics"	Paul Krugman	Textbook	14
B005	"Educated"	Tara Westover	Book	14
B006	"Becoming"	Michelle Obama	Book	14
D001	"The Godfather"	Francis Ford Coppola	Multimedia	1
O001	"The Elder Wand"	-	Other	1
O002	"The Invisibility Cloak"	-	Other	1
O003	"The Resurrection Stone"	-	Other	1

3. Loans Table:

This table stores the loan transactions, where each loan is uniquely identified by LoanID, and UserID and ItemID serve as foreign keys to the Users and Items tables.

LoanID	UserID	ItemID	LoanDate	ReturnDate
L001	S001	B001	2024-10-01	2024-10-15
L002	S001	B002	2024-10-01	2024-10-15
L003	S001	T001	2024-10-02	2024-10-16
L004	S002	B003	2024-10-03	2024-10-17
L005	S002	B004	2024-10-03	2024-10-17
L006	S002	T002	2024-10-04	2024-10-18
L007	E001	B005	2024-10-05	2024-10-19
L008	E001	B006	2024-10-05	2024-10-19
L009	E001	D001	2024-10-05	2024-10-06
L010	E001	T001	2024-10-06	2024-10-20
L011	E001	T002	2024-10-06	2024-10-20
L012	E002	O001	2024-10-01	2024-10-02
L013	E002	O002	2024-10-01	2024-10-02
L014	E002	O003	2024-10-01	2024-10-02

Benefits of 2NF:

1. **Elimination of Partial Dependencies**:
    - By moving the attributes that depend only on part of the composite key (UserID or ItemID) into their own tables, we've eliminated partial dependencies.
    - Now, Users and Items have their own independent tables, and the Loans table focuses only on the loan relationships.
2. **Reduced Redundancy**:
    - User and item information is no longer duplicated across multiple loan records. If you need to update a user’s name or the title of a book, you only need to do it in one place (in the Users or Items table), not in multiple loan records.

3. **Improved Data Integrity**:
    - By separating users, items, and loans, we reduce the chance of inconsistencies, such as a user's name being entered differently in different loan records.

Summary of 2NF Transformation:

To move the database to **2NF**, we:

- **Created a Users table** to store user-related information.
- **Created an Items table** to store item-related information.
- **Simplified the Loans table** to contain only the loan-specific data, linking to the Users and Items tables via foreign keys (UserID and ItemID).

This structure sets the stage for moving to **Third Normal Form (3NF)**, which will involve removing any transitive dependencies.

With 2NF we get the concept of primary and foreign keys. So each table needs a better description with keys.

1. Users Table

This table stores user-related information, with `UserID` as the primary key.

Column	Type	Description
UserID	Primary Key	Unique identifier for each user (Student/Employee).
FirstName		User's first name.
LastName		User's last name.
UserType		Type of user (e.g., Student or Employee).
MaxLoans		Maximum number of items this user can borrow.

Primary Key:

- **UserID** (unique for each user)

Foreign Keys:

- **None** (this is a standalone table)

2. Items Table

This table stores item-related information, with `ItemID` as the primary key.

Column	Type	Description
ItemID	Primary Key	Unique identifier for each item (book, textbook, multimedia, other).
Title		Title of the item.
Author/Creator		Author or creator of the item.
ItemType		Type of item (e.g., Book, Multimedia).
LoanDuration		Standard loan duration for this item.

Primary Key:

- **ItemID** (unique for each item)

Foreign Keys:

- **None** (this is a standalone table)

3. Loans Table

This table stores loan transaction information, with `LoanID` as the primary key. It links users to items through foreign keys.

Column	Type	Description
LoanID	Primary Key	Unique identifier for each loan transaction.
UserID	Foreign Key	References `UserID` in the `Users` table (the user who borrowed the item).
ItemID	Foreign Key	References `ItemID` in the `Items` table (the item that is being borrowed).
LoanDate		The date the item was borrowed.
ReturnDate		The date the item is due to be returned or was returned.

Primary Key:

- **LoanID** (unique for each loan transaction)

Foreign Keys:

- **UserID** (references the primary key in the `Users` table)
- **ItemID** (references the primary key in the `Items` table)

Note: This way of describing a table and its use is **the most essential tool in documenting your work in database systems**. Many other methods exist, but these descriptions are required, whereas other methods are supplementary to these descriptions.

Exercise: Library operations in 2NF (with supplementary SQL statements)

1. Adding a New User

When a new user (student or employee) registers with the library, their information (first name, last name, user type, and max loans) is added to the **Users** table.

Example Operation:

- **Insert Query** to add a new user.

  INSERT INTO Users (UserID, FirstName, LastName, UserType, MaxLoans) VALUES ('S003', 'Elif', 'Öztürk', 'Student', 5);

UserID is the **primary key**, ensuring that each user is uniquely identified. No foreign key is used for this operation since the Users table is independent.

2. Adding a New Item to the Library

When a new book, multimedia item, or other resource is added to the library, it is recorded in the **Items** table with its details (title, author/creator, item type, and loan duration).

Example Operation:

- **Insert Query** to add a new item

  INSERT INTO Items (ItemID, Title, Author/Creator, ItemType, LoanDuration) VALUES ('B007', 'The Catcher in the Rye', 'J.D. Salinger', 'Book', 14);

ItemID is the **primary key**, ensuring that each item is uniquely identified. No foreign key is involved in this operation because the Items table is independent.

3. Processing a New Loan

When a user borrows an item, a new record is added to the **Loans** table, linking the **UserID** and **ItemID** via foreign keys. The system checks that the user is within their loan limit (MaxLoans from the Users table) and that the item is available (i.e., not currently loaned).

Example Operation:

- **Insert Query** to record a new loan.

  INSERT INTO Loans (LoanID, UserID, ItemID, LoanDate, ReturnDate) VALUES ('L015', 'S001', 'B007', '2024-10-24', '2024-11-07');

LoanID is the **primary key** for this table, uniquely identifying the loan. **UserID** is a **foreign key** that references the Users table to ensure that the user exists. **ItemID** is a **foreign key** that references the Items table to ensure the item exists.

Before processing this loan:

- The system will check the Users table using the **UserID** to ensure the user hasn't exceeded their **MaxLoans**.
- The system will check the Loans table using the **ItemID** to ensure that the item isn't already on loan.

4. Returning an Item

When a user returns an item, the corresponding entry in the **Loans** table is updated with the actual return date. This operation allows the library to track the loan duration and check for any overdue items.

Example Operation:

- **Update Query** to mark an item as returned.

  UPDATE Loans SET ReturnDate = '2024-11-01' WHERE LoanID = 'L015';

LoanID (primary key) is used to uniquely identify the loan transaction to be updated. The system can also use **UserID** and **ItemID** as foreign keys to verify which user borrowed the item.

5. Tracking Overdue Items

The library can run queries to identify overdue items based on the LoanDate, ReturnDate, and LoanDuration stored in the **Loans** and **Items** tables. For overdue items, the current date exceeds the expected return date.

Example Operation:

- **Select Query** to find overdue items.

  SELECT Loans.LoanID, Users.FirstName, Users.LastName, Items.Title, Loans.LoanDate, Loans.ReturnDate FROM Loans JOIN Users ON Loans.UserID = Users.UserID JOIN Items

```
ON Loans.ItemID = Items.ItemID WHERE Loans.ReturnDate IS NULL AND
DATE_ADD(Loans.LoanDate, INTERVAL Items.LoanDuration DAY) < CURDATE();
```

This query uses **foreign keys** (UserID and ItemID) to join the Loans table with the Users and Items tables. The system calculates whether the loan is overdue by adding the **LoanDuration** from the Items table to the **LoanDate** in the Loans table and comparing it to the current date

6. Managing User Restrictions

Since students and employees have different borrowing limits (tracked by the MaxLoans field in the Users table), the system can enforce these limits when a new loan is attempted.

Example Operation:

- **Select Query** to count current loans and check the user's loan limit.

```
SELECT COUNT(*) AS LoanCount FROM Loans WHERE UserID = 'S001' AND
ReturnDate IS NULL;
```

This query counts how many items the user currently has on loan (those without a return date). The system will compare this count to the **MaxLoans** value in the Users table (retrieved by **UserID**) to determine if the user can borrow more items.

7. Managing Inventory (Checking Item Availability)

To check whether a specific item (e.g., a book or DVD) is available for loan, the system can query the **Loans** table for active loans and determine if all copies of the item are currently borrowed.

Example Operation:

- **Select Query** to check item availability.

```
SELECT COUNT(*) AS OnLoan FROM Loans WHERE ItemID = 'B001' AND ReturnDate IS
NULL;
```

This query counts how many copies of a specific item (identified by **ItemID**) are currently on loan (i.e., have no return date). The system can compare this count to the number of copies available in the Items table (if multiple copies exist).

How the Keys are Used:

- **Primary Keys** ensure that each entity (user, item, loan) is uniquely identified within its respective table. For example, UserID in the Users table, ItemID in the Items table, and LoanID in the Loans table.
- **Foreign Keys** enforce relationships between tables. For instance, the Loans table references UserID from the Users table and ItemID from the Items table, ensuring that only valid users can borrow valid items.

Summary of Key Operations and Key Usage:

- **Insertions and Updates**: Primary keys ensure unique identification of users, items, and loans.
- **Queries and Joins**: Foreign keys are used to link tables when performing multi-table queries, ensuring data integrity and consistency (e.g., linking loans to users and items).
- **Constraints and Rules**: The system enforces borrowing limits and checks availability using foreign keys and primary keys to maintain the integrity of library operations.

SELF-TEST QUIZ (WITH ANSWERS)

1. Which of the following best defines **atomicity** in a database?

A. A field contains the same data across all rows.
B. A field contains a single, indivisible value in each row.
C. A table contains multiple rows for a single record.
D. A field contains multiple values separated by commas.

Answer: B

2. True or False: In a table, each row represents a unique record, and each column represents a specific attribute of that record.

Answer: True

3. Which of the following is a characteristic of **First Normal Form (1NF)**?

A. The table must not contain any foreign keys.
B. The table contains no repeating groups, and all columns contain atomic values.
C. All non-primary key attributes must depend on the entire primary key.
D. The table has no primary key.

Answer: B

4. True or False: A **primary key** in a table must contain unique values and cannot contain NULL values.

Answer: True

5. What is the main goal of **Second Normal Form (2NF)**?

A. To remove all redundancies from the database.
B. To ensure every non-primary key attribute depends on the entire primary key.
C. To eliminate all foreign keys.
D. To make every column atomic.

Answer: B

6. Which of the following statements about **foreign keys** is correct?

A. A foreign key always points to a unique record in another table.
B. A foreign key ensures that a value in a table's column matches the primary key in another table.
C. A foreign key must always contain NULL values.
D. A foreign key can only be used in 2NF or higher tables.

Answer: B

7. True or False: In 2NF, partial dependencies occur when a non-primary key attribute depends on part of a composite primary key.

Answer: True

8. Which of the following operations would likely lead to **data redundancy** in a table?

A. Storing a user's ID and name in one table and their loan history in another.
B. Storing a book's title, author, and loan status in the same table as every loan transaction.
C. Using a foreign key to link loan records to a user's ID in a separate table.
D. Using a primary key to uniquely identify each loan transaction.

Answer: B

9. Which of the following is an example of a **composite key**?

A. A single column that contains unique values in each row.
B. Two or more columns that together uniquely identify each row in the table.
C. A column that references another table's primary key.
D. A field containing multiple values.

Answer: B

10. True or False: A **foreign key** creates a relationship between two tables by linking one table's column to the primary key of another table.

Answer: True

POST-QUIZ REVIEW

Atomicity (Q1) - Understanding that each field should contain a single, indivisible value.

Tables and Rows (Q2) - Clarifying the relationship between rows (records) and columns (attributes).

First Normal Form (1NF) (Q3) - Eliminating repeating groups and ensuring atomicity.

Primary Keys (Q4) - Unique and non-null identifiers for each row.

Second Normal Form (2NF) (Q5, Q7) - Elimination of partial dependencies.

Foreign Keys (Q6, Q10) - Creating relationships between tables.

Data Redundancy (Q8) - Identifying how poor design can lead to redundant data.

Composite Keys (Q9) - Using multiple columns together to create a unique key.

Adding Auto-Incrementing IDs

Benefits of Adding Auto-Incrementing IDs:

1. **Simplified Key Construction:**
    - Auto-incrementing IDs provide a simple, numeric way to uniquely identify each record in a table.
    - This avoids the need for composite keys (which are more complex) and can simplify foreign key relationships, making queries more efficient and easier to write.
2. **Improved Indexing:**
    - Indexes are crucial for improving database performance, especially when tables grow large.
    - Using an auto-incrementing integer as the primary key provides a small, sequential, and fixed-size index that is easy for the database to manage.
    - It improves look-up times since integer indexes are typically faster to search than composite or string-based keys.
3. **Better Data Integrity and Maintenance:**
    - With an auto-incrementing ID, you reduce the chances of errors caused by human entry when dealing with complex or composite keys (like UserID combined with another column).
    - It also ensures that each row is always uniquely identifiable with a numeric ID, **even if other attributes (like UserID or ItemID) change in the future.**

1. Users Table (Adding UserAutoID)

Currently, UserID is the primary key, which might be a string (e.g., 'S001', 'E001'). By adding an auto-incrementing column, such as UserAutoID, we simplify key usage and ensure numeric indexing.

UserAutoID	UserID	FirstName	LastName	UserType	MaxLoans
1	S001	Ayşe	Yılmaz	Student	5
2	S002	Mehmet	Kaya	Student	5
3	E001	Ahmet	Demir	Employee	10
4	E002	Kerem	Aktürkoğlu	Employee	10

- **Primary Key**: UserAutoID (auto-incrementing integer)
- **Foreign Key**: UserAutoID would be referenced in the **Loans** table instead of UserID.

- This avoids having to use the UserID string as the foreign key, which can be less efficient for indexing.

2. Items Table (Adding ItemAutoID)

The Items table uses ItemID (like 'B001' or 'T001') as the primary key. By adding ItemAutoID, we simplify how we reference items in the **Loans** table.

ItemAutoID	ItemID	Title	Author/Creator	ItemType	LoanDuration
1	B001	"The Night Circus"	Erin Morgenstern	Book	14
2	B002	"The Silent Patient"	Alex Michaelides	Book	14
3	T001	"Principles of Economics"	N. Gregory Mankiw	Textbook	14
4	B003	"Where the Crawdads Sing"	Delia Owens	Book	14

- **Primary Key**: ItemAutoID (auto-incrementing integer)
- **Foreign Key**: ItemAutoID would be referenced in the **Loans** table instead of ItemID.
- This would make the foreign key in the **Loans** table an integer, which is more efficient than using string-based ItemID.

3. Loans Table (Adding LoanAutoID)

Adding an auto-incrementing primary key to the **Loans** table simplifies the construction of the primary key. Currently, LoanID might be manually assigned or constructed from other columns, but an auto-incrementing LoanAutoID would simplify this.

LoanAutoID	UserAutoID	ItemAutoID	LoanDate	ReturnDate
1	1	1	2024-10-01	2024-10-15
2	1	2	2024-10-01	2024-10-15
3	2	3	2024-10-03	2024-10-17

- **Primary Key**: LoanAutoID (auto-incrementing integer)
- **Foreign Keys**: UserAutoID and ItemAutoID reference the Users and Items tables respectively.

This design ensures each loan record has a simple, unique identifier while linking efficiently to users and items using numeric foreign keys.

The Third Normal Form (3NF)

Third Normal Form (3NF) builds upon the principles of **Second Normal Form (2NF)** by further reducing data redundancy and ensuring that every non-key attribute depends only on the primary key. The key difference between 2NF and 3NF is that **3NF eliminates transitive dependencies**, whereas 2NF focuses on eliminating partial dependencies.

Key Characteristics of 3NF:

1. **In 2NF**:
    - The table is in **First Normal Form (1NF)** (no repeating groups, atomic values).
    - The table has no **partial dependencies**, meaning every non-key attribute depends on the entire primary key (if a composite key is used).
2. **In 3NF**:
    - The table is already in 2NF.
    - **Transitive dependencies** are removed. This means that non-key attributes should not depend on other non-key attributes, but only on the primary key.

Transitive Dependency Example:

A **transitive dependency** occurs when a non-key attribute depends on another non-key attribute rather than directly on the primary key.

For instance, if you have a table where:

- ItemID is the primary key.
- AuthorName is a non-key attribute that also determines AuthorBio.

In this case, AuthorBio depends on AuthorName rather than directly on ItemID, creating a transitive dependency. This violates 3NF.

To remove this transitive dependency, you would:

- Move AuthorName and AuthorBio into a separate Authors table.
- In the original table, only keep the **foreign key** (e.g., AuthorID) that references the Authors table.

Requirements for a Table to be in 3NF:

1. The table must be in **2NF** (no partial dependencies).
2. **No transitive dependencies**: Non-key attributes must depend only on the primary key and not on any other non-key attributes.
3. **Every non-prime attribute** (attributes that are not part of any candidate key) must depend **directly and only** on the primary key.

Typical Changes to Achieve 3NF:

1. **Identify Transitive Dependencies**:
    - Find any attributes that are dependent on other non-key attributes rather than directly on the primary key.
2. **Move Dependent Data to Separate Tables**:
    - Create new tables for attributes that have transitive dependencies.
    - Add foreign keys to maintain the relationships between the original table and the new tables.
3. **Normalize Non-Key Attributes**:
    - Ensure that all non-key attributes depend directly on the primary key.

Let's consider a simplified **Books Table** in 2NF:

2NF Example (Books Table):

ItemID	Title	AuthorName	AuthorCountry	PublisherName
B001	"The Night Circus"	Erin Morgenstern	USA	Doubleday
B002	"The Silent Patient"	Alex Michaelides	UK	Celadon Books
B003	"Educated"	Tara Westover	USA	Random House

- In this table, ItemID is the primary key.
- **Transitive dependency**: AuthorCountry depends on AuthorName, not directly on ItemID. PublisherName depends on the publisher, which is also indirectly linked to ItemID.

To Convert to 3NF:

We need to remove the transitive dependencies by creating separate tables for **Authors** and **Publishers**.

Authors Table: Store AuthorName and AuthorCountry in a separate table, where AuthorID is the primary key.

AuthorID	AuthorName	AuthorCountry
A001	Erin Morgenstern	USA
A002	Alex Michaelides	UK
A003	Tara Westover	USA

Publishers Table: Store `PublisherName` in a separate table, where `PublisherID` is the primary key.

PublisherID	PublisherName
P001	Doubleday
P002	Celadon Books
P003	Random House

Revised Books Table (Now in 3NF): The Books table now references AuthorID and PublisherID instead of storing the AuthorName, AuthorCountry, and PublisherName directly. This removes the transitive dependencies.

ItemID	Title	AuthorID	PublisherID
B001	"The Night Circus"	A001	P001
B002	"The Silent Patient"	A002	P002
B003	"Educated"	A003	P003

ALL TABLES in 3NF

1. Users Table

This table stores the information about users (students and employees). We already moved user-related information into a separate table in 2NF, and there's no transitive dependency here, so this table stays the same.

UserAutoID	UserID	FirstName	LastName	UserTypeID	MaxLoans
1	S001	Ayşe	Yılmaz	1	5
2	S002	Mehmet	Kaya	1	5
3	E001	Ahmet	Demir	2	10
4	E002	Kerem	Aktürkoğlu	2	10

- **Primary Key**: UserAutoID
- **Foreign Key**: UserTypeID references the UserTypes table.

2. UserTypes Table

This table handles the user type (e.g., student, employee). UserType was a candidate for a transitive dependency, so we extract it into its own table.

UserTypeID	UserType
1	Student
2	Employee

- **Primary Key**: UserTypeID

3. Items Table

This table stores information about library items (books, textbooks, multimedia, other resources). There were no transitive dependencies in the Items table itself in 2NF, but we move item type details into their own table for 3NF.

ItemAutoID	ItemID	Title	Author/Creator	ItemTypeID	LoanDuration
1	B001	"The Night Circus"	Erin Morgenstern	1	14
2	B002	"The Silent Patient"	Alex Michaelides	1	14
3	T001	"Principles of Economics"	N. Gregory Mankiw	2	14
4	B003	"Where the Crawdads Sing"	Delia Owens	1	14
5	D001	"The Godfather"	Francis Ford Coppola	3	1
6	O001	"The Elder Wand"	-	4	1

- **Primary Key**: ItemAutoID
- **Foreign Key**: ItemTypeID references the ItemTypes table.

4. ItemTypes Table

This table handles different item types (e.g., books, textbooks, multimedia, other resources), which were potential transitive dependencies in the Items table.

ItemTypeID	ItemType
1	Book
2	Textbook
3	Multimedia
4	Other

- **Primary Key**: ItemTypeID

5. Loans Table

This table records all loan transactions, and there are no transitive dependencies here because it links directly to users and items.

LoanAutoID	UserAutoID	ItemAutoID	LoanDate	ReturnDate
1	1	1	2024-10-01	2024-10-15
2	1	2	2024-10-01	2024-10-15
3	2	3	2024-10-03	2024-10-17
4	3	4	2024-10-05	2024-10-19
5	4	6	2024-10-01	2024-10-02

- **Primary Key**: LoanAutoID
- **Foreign Keys**:
  - UserAutoID references the Users table.
  - ItemAutoID references the Items table.