Database Normalization

Use Case #1

Simple university database as an example, which includes information about students, the courses they're enrolled in, and their grades.

Original Table (Unnormalized)

Let's assume we have an unnormalized table that contains student information, their courses, and grades:

StudentID	StudentName	CourseIDs	Grades
1	Alice	[101, 102]	[A, B]
2	Bob	[101, 103]	[B, C]
3	Charlie	[102]	[B]

This table has multiple values in the `CourseIDs` and `Grades` fields for each student, which is a violation of the normal forms.

1NF (First Normal Form)

To bring the table into 1NF, we need to eliminate repeating groups and ensure each field contains only atomic values.

StudentID	StudentName	CourseID	Grade
1	Alice	101	A
1	Alice	102	B
2	Bob	101	B
2	Bob	103	C
3	Charlie	102	B

Now, each row represents a unique student-course combination, and all fields contain only single values.

2NF (Second Normal Form)

For 2NF, we need to remove partial dependencies, i.e., non-key attributes should depend on the whole primary key.

We identify two primary keys here: `StudentID` and `CourseID`. The `StudentName` is dependent only on `StudentID`, not on `CourseID`. So, we split the table to remove the partial dependency.

Students Table:

StudentID	StudentName
1	Alice
2	Bob
3	Charlie

Enrollments Table:

StudentID	CourseID	Grade
1	101	A
1	102	B
2	101	B
2	103	C
3	102	B

3NF (Third Normal Form)

In 3NF, we remove transitive dependencies; attributes should depend only on the primary key.

If we had additional information like `CourseName` that depends on `CourseID` but not on `StudentID`, we would need to create another table to eliminate this transitive dependency.

Courses Table:

CourseID	CourseName
101	Math
102	Science
103	Literature

Enrollments Table (Revised):

StudentID	CourseID	Grade
1	101	A
1	102	B
2	101	B
2	103	C
3	102	B

Now, each table is in 3NF. The `Students` table describes students, the `Courses` table lists courses, and the `Enrollments` table shows which student is enrolled in which course with their respective grades. There are no repeating groups, partial dependencies, or transitive dependencies.

Use Case #2

An online retail store's database that tracks customer orders.

Original Table (Unnormalized)

In this unnormalized table, each row contains information about a customer, their orders, and the products in those orders.

CustomerID	CustomerName	Orders	Products
101	John Doe	[Order001, Order005]	[Book, Pen, Notebook]
102	Jane Smith	[Order002]	[Pencil, Eraser]
103	Emily Jones	[Order003, Order004]	[Backpack, Water Bottle]

This table has multiple values in the `Orders` and `Products` fields for each customer, violating the principles of normalization.

1NF (First Normal Form)

In 1NF, we need to eliminate repeating groups and ensure each field contains atomic values.

CustomerID	CustomerName	OrderID	Product
101	John Doe	Order001	Book
101	John Doe	Order001	Pen
101	John Doe	Order005	Notebook
102	Jane Smith	Order002	Pencil
102	Jane Smith	Order002	Eraser
103	Emily Jones	Order003	Backpack
103	Emily Jones	Order004	Water Bottle

Each row now represents a unique customer-order-product combination, with all fields containing single values.

2NF (Second Normal Form)

For 2NF, we remove partial dependencies, so non-key attributes should depend on the whole primary key.

We identify two primary keys here: `CustomerID` and `OrderID`. The `CustomerName` is dependent only on `CustomerID`, not on `OrderID`. Therefore, we split the table to remove partial dependency.

Customers Table:

CustomerID	CustomerName
101	John Doe
102	Jane Smith
103	Emily Jones

Orders Table:

CustomerID	OrderID	Product
101	Order001	Book
101	Order001	Pen
101	Order005	Notebook
102	Order002	Pencil
102	Order002	Eraser
103	Order003	Backpack
103	Order004	Water Bottle

3NF (Third Normal Form)

In 3NF, we remove transitive dependencies; attributes should depend only on the primary key.

If we had additional information like `ProductPrice` that depends on `Product` but not directly on `CustomerID` or `OrderID`, we would need to create another table to eliminate this transitive dependency.

Products Table:

ProductID	Product	Price
Prod001	Book	$15
Prod002	Pen	$3
Prod003	Notebook	$7
Prod004	Pencil	$2
Prod005	Eraser	$1
Prod006	Backpack	$20
Prod007	Water Bottle	$8

Orders Table (Revised):

CustomerID	OrderID	ProductID
101	Order001	Prod001
101	Order001	Prod002
101	Order005	Prod003
102	Order002	Prod004
102	Order002	Prod005
103	Order003	Prod006
103	Order004	Prod007

Now, the `Customers` table defines customers, the `Products` table lists products with their prices, and the `Orders` table shows which customer placed which order for specific products. This setup adheres to the principles of 1NF, 2NF, and 3NF, ensuring data integrity and reducing redundancy.