Database Normalization

> *Refer to this article for more in-depth understanding of Database Normalization: Link*

Introduction to Normalization

Database normalization is a systematic, iterative process of organizing data in a relational database. It involves dividing larger tables into smaller, well-structured tables and defining relationships between them to reduce redundancy and improve data integrity.

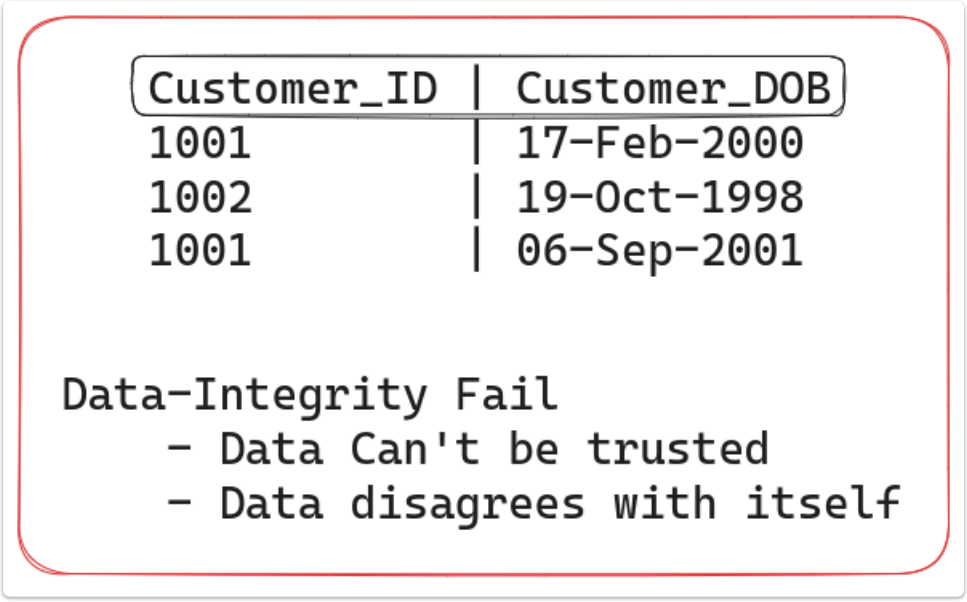
Normalization follows a series of progressive "normal forms," with each form building upon the previous one. In this lecture, we'll explore the first three normal forms (1NF, 2NF, and 3NF) with practical MySQL examples.

First Normal Form (1NF)

Definition

A table is in First Normal Form (1NF) when:

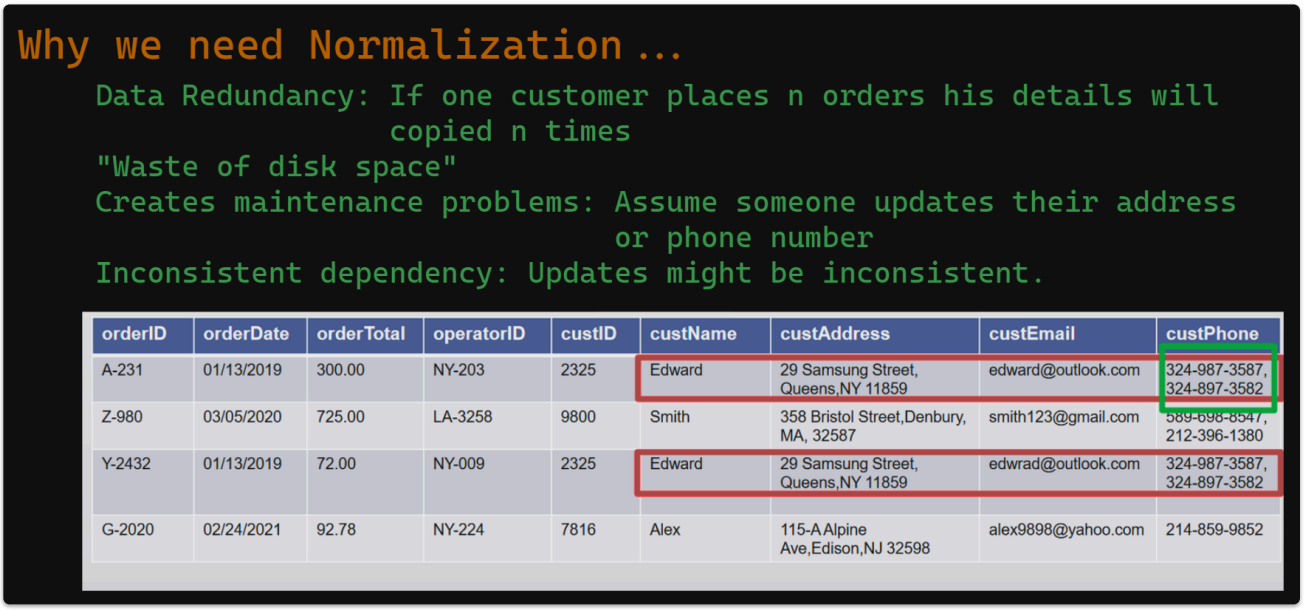
1. Using row order to convey information is not permitted
2. Mixing data types within the same column is not permitted
3. Having a table without a primary key is not permitted
4. Repeating groups are not permitted



Normalized Tables are:

1. Easier to understand.
2. Easier to enhance and extend
3. Protected from:
    1. Insertion Anomalies
    2. Update Anomalies

3. Deletion Anomalies



orderID	orderDate	orderTotal	operatorID	custID	custName	custAddress	custEmail	custPhone
A-231	01/13/2019	300.00	NY-203	2325	Edward	29 Samsung Street, Queens,NY 11859	edward@outlook.com	324-987-3587, 324-897-3582
Z-980	03/05/2020	725.00	LA-3258	9800	Smith	358 Bristol Street,Denbury, MA, 32587	smith123@gmail.com	589-698-8547, 212-396-1380
Y-2432	01/13/2019	72.00	NY-009	2325	Edward	29 Samsung Street, Queens,NY 11859	edwrad@outlook.com	324-987-3587, 324-897-3582
G-2020	02/24/2021	92.78	NY-224	7816	Alex	115-A Alpine Ave,Edison,NJ 32598	alex9898@yahoo.com	214-859-9852

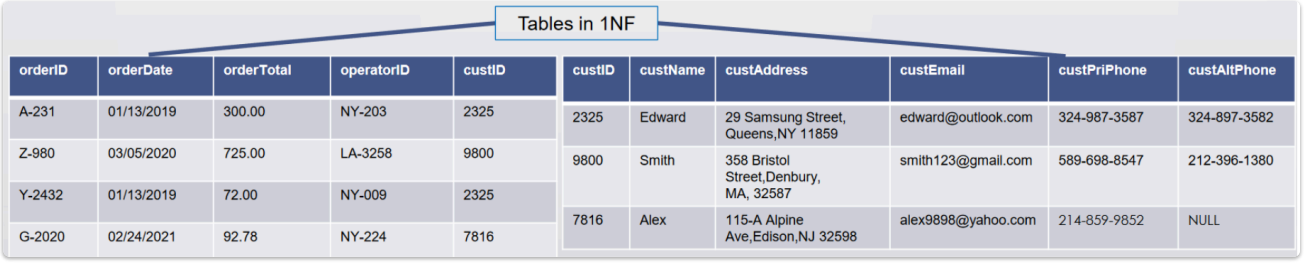
Problems Without 1NF

orderID	orderDate	orderTotal	operatorID	custID	custName	custAddress	custEmail	custPhone
A-231	01/13/2019	300.00	NY-203	2325	Edward	29 Samsung Street, Queens,NY 11859	edward@outlook.com	324-987-3587, 324-897-3582
Z-980	03/05/2020	725.00	LA-3258	9800	Smith	358 Bristol Street,Denbury, MA, 32587	smith123@gmail.com	589-698-8547, 212-396-1380
Y-2432	01/13/2019	72.00	NY-009	2325	Edward	29 Samsung Street, Queens,NY 11859	edwrad@outlook.com	324-987-3587, 324-897-3582
G-2020	02/24/2021	92.78	NY-224	7816	Alex	115-A Alpine Ave,Edison,NJ 32598	alex9898@yahoo.com	214-859-9852

Problems:

- Customer Edward (custID 2325) appears twice with duplicate information
- Smith has multiple phone numbers in a single cell (mixing data types)
- The same primary key (custID) appears multiple times

Example: Applying 1NF



orderID	orderDate	orderTotal	operatorID	custID
A-231	01/13/2019	300.00	NY-203	2325
Z-980	03/05/2020	725.00	LA-3258	9800
Y-2432	01/13/2019	72.00	NY-009	2325
G-2020	02/24/2021	92.78	NY-224	7816

custID	custName	custAddress	custEmail	custPriPhone	custAltPhone
2325	Edward	29 Samsung Street, Queens,NY 11859	edward@outlook.com	324-987-3587	324-897-3582
9800	Smith	358 Bristol Street,Denbury, MA, 32587	smith123@gmail.com	589-698-8547	212-396-1380
7816	Alex	115-A Alpine Ave,Edison,NJ 32598	alex9898@yahoo.com	214-859-9852	NULL

1. Separating phone numbers into different columns (custPriPhone and custAltPhone)
2. Ensuring each column contains only one type of data
3. Making sure we have a proper primary key

Original Table (Not in 1NF):

```
CREATE TABLE orders_not_normalized (
    orderID VARCHAR(10),
    orderDate DATE,
```

```
    orderTotal DECIMAL(10,2),
    operatorID VARCHAR(10),
    custID INT,
    custName VARCHAR(50),
    custAddress VARCHAR(100),
    custEmail VARCHAR(50),
    custPhone VARCHAR(100) -- Contains multiple phone numbers!
);
```

After 1NF:

```
CREATE TABLE orders_1nf (
    orderID VARCHAR(10) PRIMARY KEY,
    orderDate DATE,
    orderTotal DECIMAL(10,2),
    operatorID VARCHAR(10),
    custID INT,
    custName VARCHAR(50),
    custAddress VARCHAR(100),
    custEmail VARCHAR(50),
    custPriPhone VARCHAR(15),
    custAltPhone VARCHAR(15)
);
```

Second Normal Form (2NF)

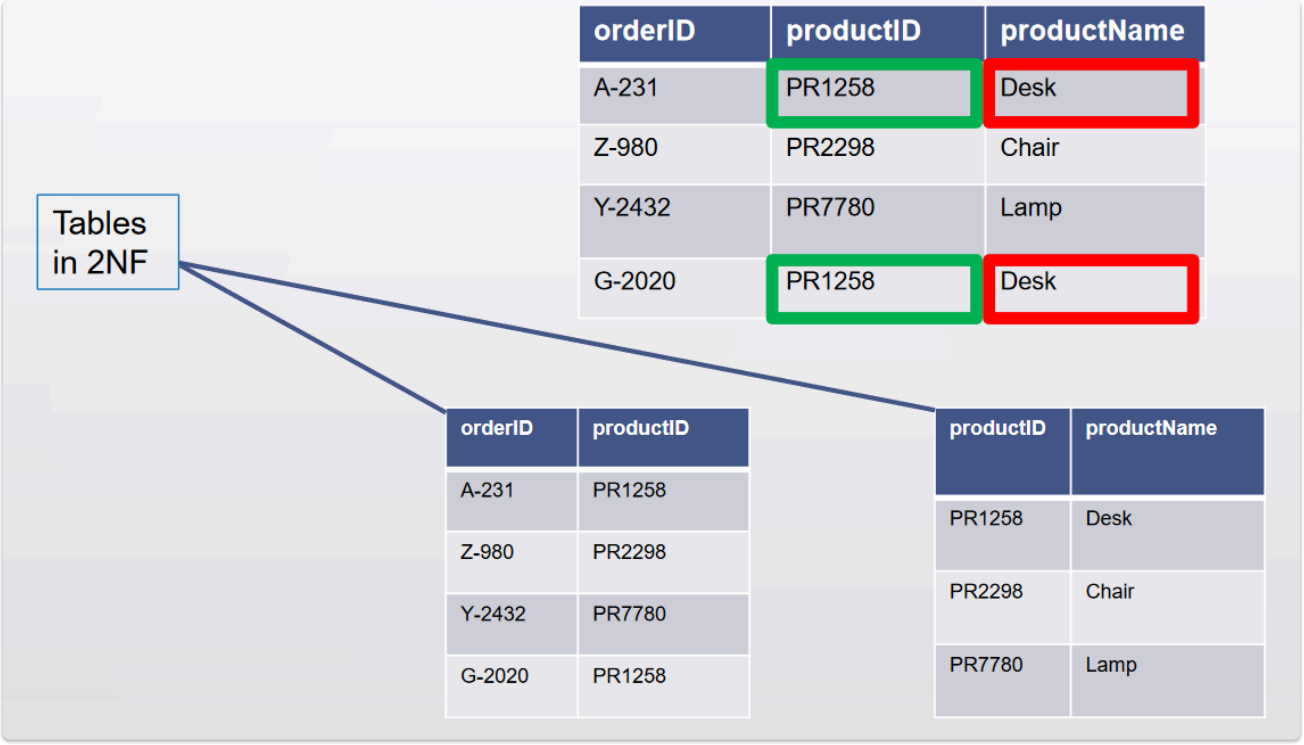
Definition

A table is in Second Normal Form (2NF) when:

1. It is already in 1NF
2. All non-key attributes are fully functionally dependent on the primary key

In simpler terms, 2NF removes partial dependencies. If a non-key column depends on only part of a composite primary key, it should be moved to a separate table.

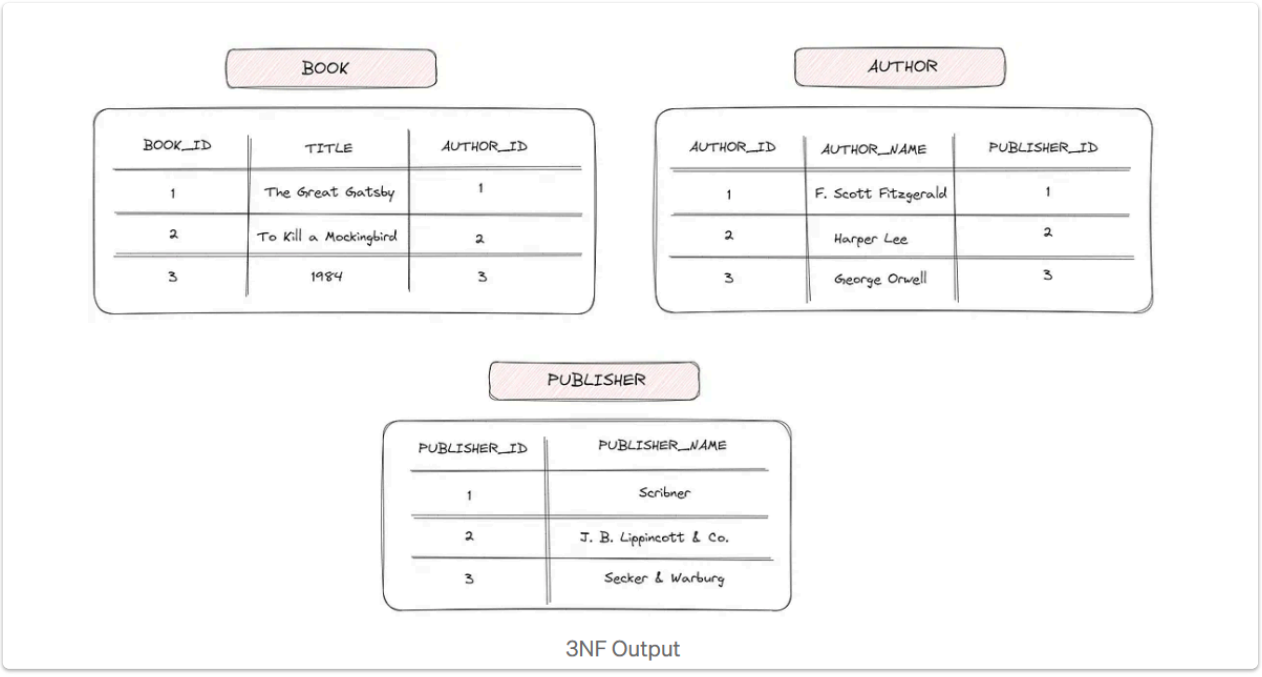
Problems Without 2NF



- Product information (productID, productName) is repeated for different orders
- Product details depend only on productID, not on the entire primary key (orderID + productID)

> *Let's take another example. Consider a table that stores information about books. The table might have columns like `book_id`, `title`, `author`, and `publisher`.*

> *However, the `publisher` column could be dependent on the `author` column, rather than on the `book_id` column. To bring this table to 3NF, we need to split it into two tables, one for book information and one for author information.*



3NF Output

Example: Applying 2NF

To achieve 2NF, we split the table into separate tables:

Orders-Products Table:

```
CREATE TABLE order_products (
    orderID VARCHAR(10),
    productID VARCHAR(10),
    PRIMARY KEY (orderID, productID),
    FOREIGN KEY (orderID) REFERENCES orders(orderID),
    FOREIGN KEY (productID) REFERENCES products(productID)
);
```

Products Table:

```
CREATE TABLE products (
    productID VARCHAR(10) PRIMARY KEY,
    productName VARCHAR(50)
);
```

This way, product information is stored only once and referenced as needed, eliminating redundancy.

Third Normal Form (3NF)

Definition

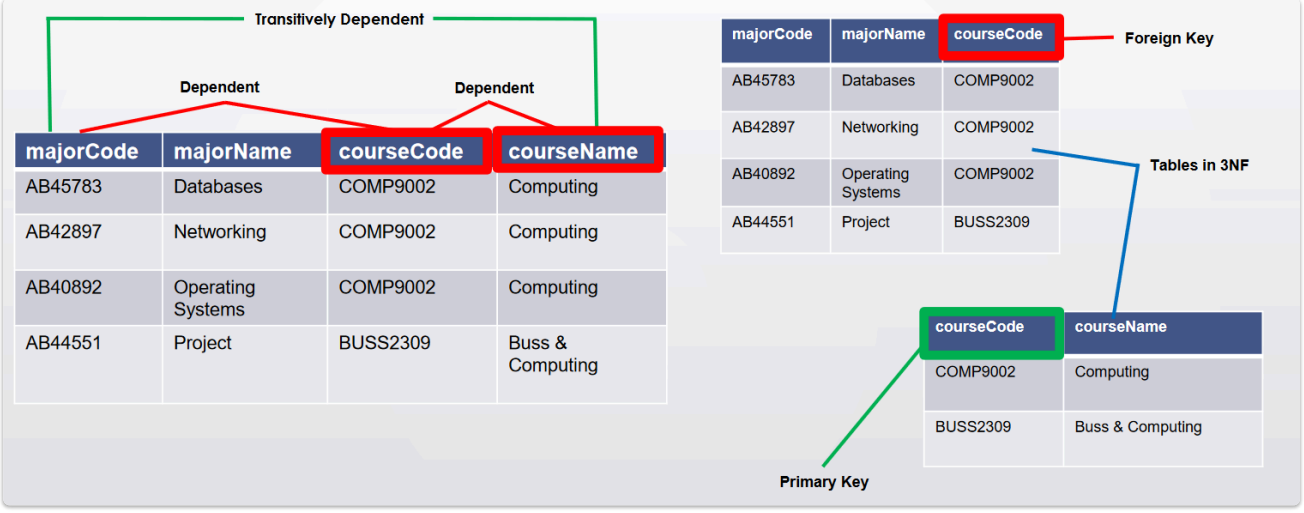
A table is in Third Normal Form (3NF) when:

1. It is already in 2NF
2. All non-key attributes are non-transitively dependent on the primary key

> *Every non-key attribute in a table should depend on the key, the whole key and nothing but the key.*

In other words, non-key columns should not depend on other non-key columns.

Problems Without 3NF



- courseName depends on courseCode, not directly on majorCode (transitive dependency)

- If courseCode changes, we'd need to update multiple rows

Example: Applying 3NF

Create a separate table for courses

Majors Table:

```
CREATE TABLE majors (
    majorCode VARCHAR(10) PRIMARY KEY,
    majorName VARCHAR(50),
    courseCode VARCHAR(10),
    FOREIGN KEY (courseCode) REFERENCES courses(courseCode)
);
```

Courses Table:

```
CREATE TABLE courses (
    courseCode VARCHAR(10) PRIMARY KEY,
    courseName VARCHAR(50)
);
```

Now courseName depends directly on the primary key courseCode, eliminating transitive dependencies.

Boyce-Codd Normal Form (BCNF):

- BCNF is a stricter form of 3NF, where every determinant is a candidate key.
- Example: In a table containing details of a university course, if the combination of `course_code` and `semester` uniquely determines `instructor`, `room_number`, and `meeting_time`, it is not in BCNF. To achieve BCNF, the table must be split into multiple tables.

Fourth Normal Form (4NF) and Fifth Normal Form (5NF) focus on multi-valued dependencies and join dependencies, which are more advanced and typically not as commonly encountered in standard database design.

Complete Example: Normalization Process

Let's walk through a complete example, transforming a denormalized table step by step.

Original Denormalized Table

```
CREATE TABLE student_enrollments (
    student_id INT,
    student_name VARCHAR(100),
    student_email VARCHAR(100),
    course_id VARCHAR(10),
    course_name VARCHAR(100),
    instructor_id VARCHAR(10),
    instructor_name VARCHAR(100),
```

```
    instructor_dept VARCHAR(50),
    enrollment_date DATE,
    grade VARCHAR(2)
);

INSERT INTO student_enrollments VALUES
(1001, 'John Smith', 'john@email.com', 'CS101', 'Intro to Programming', 'INS01', 'Dr.
Brown', 'Computer Science', '2023-01-15', 'A'),
(1001, 'John Smith', 'john@email.com', 'MATH201', 'Calculus', 'INS02', 'Dr. Davis',
'Mathematics', '2023-01-16', 'B+'),
(1002, 'Emily Jones', 'emily@email.com', 'CS101', 'Intro to Programming', 'INS01',
'Dr. Brown', 'Computer Science', '2023-01-15', 'A-');
```

Step 1: Apply 1NF

Ensure no repeating groups, establish a primary key.

```
-- Already satisfied in this case, but we'll explicitly set a composite primary key
CREATE TABLE student_enrollments_1nf (
    student_id INT,
    course_id VARCHAR(10),
    student_name VARCHAR(100),
    student_email VARCHAR(100),
    course_name VARCHAR(100),
    instructor_id VARCHAR(10),
    instructor_name VARCHAR(100),
    instructor_dept VARCHAR(50),
    enrollment_date DATE,
    grade VARCHAR(2),
    PRIMARY KEY (student_id, course_id)
);
```

Step 2: Apply 2NF

Remove partial dependencies.

```
-- Students table
CREATE TABLE students_2nf (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100),
    student_email VARCHAR(100)
);

-- Courses table
CREATE TABLE courses_2nf (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100),
    instructor_id VARCHAR(10),
    instructor_name VARCHAR(100),
    instructor_dept VARCHAR(50)
);
```

```
-- Enrollments table (junction table)
CREATE TABLE enrollments_2nf (
    student_id INT,
    course_id VARCHAR(10),
    enrollment_date DATE,
    grade VARCHAR(2),
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students_2nf(student_id),
    FOREIGN KEY (course_id) REFERENCES courses_2nf(course_id)
);
```

Step 3: Apply 3NF

Remove transitive dependencies.

```
-- Students table
CREATE TABLE students_3nf (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100),
    student_email VARCHAR(100)
);

-- Instructors table
CREATE TABLE instructors_3nf (
    instructor_id VARCHAR(10) PRIMARY KEY,
    instructor_name VARCHAR(100),
    instructor_dept VARCHAR(50)
);

-- Courses table
CREATE TABLE courses_3nf (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100),
    instructor_id VARCHAR(10),
    FOREIGN KEY (instructor_id) REFERENCES instructors_3nf(instructor_id)
);

-- Enrollments table (junction table)
CREATE TABLE enrollments_3nf (
    student_id INT,
    course_id VARCHAR(10),
    enrollment_date DATE,
    grade VARCHAR(2),
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students_3nf(student_id),
    FOREIGN KEY (course_id) REFERENCES courses_3nf(course_id)
);
```

Benefits of Normalization

1. **Elimination of Redundancy**: Less duplicate data means less storage space and easier maintenance.
2. **Data Integrity**: Less redundancy means fewer chances for inconsistencies.
3. **Flexibility**: Easier to modify and extend the database structure.
4. **Better Query Performance**: While normalization might slow down some queries, it generally improves performance for complex operations.

Common MySQL Commands for Managing Normalized Tables

Creating Foreign Keys

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer
FOREIGN KEY (customer_id) REFERENCES customers(id);
```

Joining Normalized Tables

```
-- Get order details with customer information
SELECT o.order_id, o.order_date, c.customer_name, c.email
FROM orders o
JOIN customers c ON o.customer_id = c.id;

-- Get complete order information with products
SELECT o.order_id, o.order_date, c.customer_name,
       p.product_name, op.quantity, p.price
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN order_products op ON o.order_id = op.order_id
JOIN products p ON op.product_id = p.id;
```

Practical Considerations

When to Denormalize

Sometimes, denormalization (intentionally adding redundancy) is appropriate:

- For read-heavy applications
- When query performance is critical
- For data warehousing and reporting systems

```
-- Denormalized table for reporting
CREATE TABLE order_report (
    order_id VARCHAR(10),
    order_date DATE,
    customer_name VARCHAR(100),
    customer_email VARCHAR(100),
    product_name VARCHAR(100),
    quantity INT,
    price DECIMAL(10,2),
```

```
    total_price DECIMAL(10,2)
);
```

Indexes for Normalized Databases

Proper indexing is crucial for maintaining performance in normalized databases:

```
-- Add index to foreign key columns
CREATE INDEX idx_customer_id ON orders(customer_id);
CREATE INDEX idx_order_id ON order_products(order_id);
CREATE INDEX idx_product_id ON order_products(product_id);
```

Conclusion

Normalization is a fundamental concept in relational database design that helps maintain data integrity and reduce redundancy. By following the progressive normal forms (1NF, 2NF, 3NF), we can create well-structured databases that are efficient, flexible, and easier to maintain.

Remember that normalization is not always about reaching the highest normal form possible. The appropriate level of normalization depends on your specific application requirements, balancing data integrity with performance considerations.