In SQL, normalization refers to the process of organizing data in a database to reduce redundancy and improve data integrity. There are different normal forms, each with its own set of rules. Here, I'll provide examples for the first three normal forms: 1NF, 2NF, and 3NF.

1. First Normal Form (1NF):

1NF ensures that a table has no repeating groups of columns. Each column contains atomic values, and there are no repeating groups or arrays.

Example:

Consider the following table that violates 1NF:

```sql
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50),
    Skills VARCHAR(255)
);

INSERT INTO Employee VALUES (1, 'John Doe', 'Java, SQL, Python');

Employes

Empid  empname skills
1         johndoe ,sql,
1         johndoe java
1         johndoe python
```

This violates 1NF because the "Skills" column contains multiple values. To normalize it, we create a separate table for skills:

```sql
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50)
);

CREATE TABLE EmployeeSkills (
    EmployeeID INT,
    Skill VARCHAR(50),
    PRIMARY KEY (EmployeeID, Skill),
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
```

```
);

INSERT INTO Employee VALUES (1, 'John Doe');
INSERT INTO EmployeeSkills VALUES (1, 'Java');
INSERT INTO EmployeeSkills VALUES (1, 'SQL');
INSERT INTO EmployeeSkills VALUES (1, 'Python');
```

2. Second Normal Form (2NF):

2NF builds on 1NF and eliminates partial dependencies. In a 2NF table, all non-prime attributes must be fully functionally dependent on the primary key.

Example:

Consider a table that violates 2NF:

```sql
CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    ProductName VARCHAR(50),
    Quantity INT,
    PRIMARY KEY (OrderID, ProductID)
);
```

In this case, "ProductName" is dependent on "ProductID," which is a part of the composite primary key. To normalize it:

```sql
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(50)
);

CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    PRIMARY KEY (OrderID, ProductID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

3. Third Normal Form (3NF):

3NF eliminates transitive dependencies. In a 3NF table, no non-prime attribute should be transitively dependent on the primary key.

Example:

Consider a table that violates 3NF:

```sql
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50),
    Department VARCHAR(50),
    ManagerName VARCHAR(50)
);
```

Here, "ManagerName" is transitively dependent on "EmployeeID" through "Department." To normalize it:

```sql
CREATE TABLE Department (
    DepartmentName VARCHAR(50) PRIMARY KEY,
    ManagerName VARCHAR(50)
);

CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50),
    DepartmentName VARCHAR(50),
    FOREIGN KEY (DepartmentName) REFERENCES Department(DepartmentName)
);
```

These examples illustrate the process of normalization in SQL up to the third normal form. Depending on the specific requirements and complexity of your data, you might need to go further and apply higher normal forms.

##

1. **Elimination of Data Redundancy:**
   - **Example:**
     Suppose you have a table that stores customer information along with the city and state. Without normalization, the same city and state information may be repeated for multiple customers, leading to redundancy.
     ```sql
     CustomerID | CustomerName | City       | State
     ----------------------------------------------
     1          | John Doe     | New York   | NY,Austin
     2          | Jane Smith   | New York   | NY
     ```
     After normalization, the city and state information can be stored in a separate table to avoid redundancy.

2. **Data Consistency:**
   - **Example:**
     In a denormalized structure, if you update the city or state for one customer, you must ensure that the same update is applied to all occurrences of that customer's information. In a normalized structure, updates are made in one place, ensuring consistency.

3. **Improved Data Integrity:**
   - **Example:**
     In a normalized structure, foreign key constraints can be used to maintain referential integrity. For instance, if you have an Orders table referencing a Customers table, the foreign key ensures that an order is associated with an existing customer.
     ```sql
     CREATE TABLE Orders (
         OrderID INT PRIMARY KEY,
         CustomerID INT,
         OrderDate DATE,
         FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
     );
     ```

4. **Simplified Updates:**
   - **Example:**
     In a denormalized structure, updating information may require changes in multiple places. In a normalized structure, updates are localized to specific tables, making the process more straightforward and reducing the likelihood of errors.

5. **Enhanced Query Performance:**
   - **Example:**
     Normalization can lead to more efficient querying because data is stored in a way that reflects relationships. For instance, if you want to find all employees with a specific skill in a

normalized structure, you can query the EmployeeSkills table directly, avoiding the need to parse a delimited string as you might in a denormalized structure.

It's important to note that while normalization has these advantages, there are also trade-offs, such as increased complexity of queries and potential performance overhead for certain types of reads. The level of normalization should be chosen based on the specific requirements and usage patterns of the database.