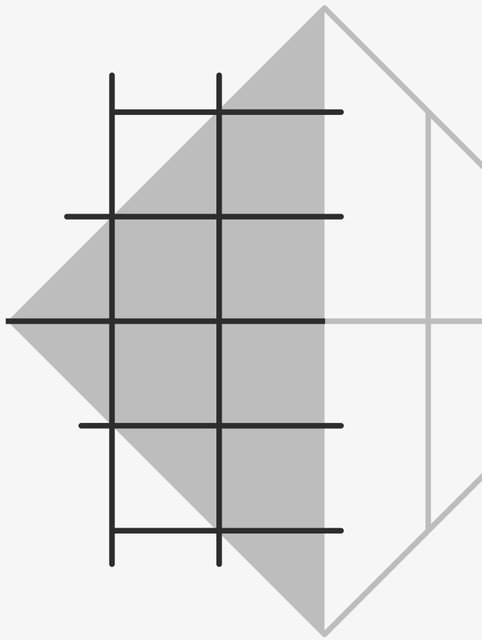
Normalization

Shrajan Shukla

What is Normalization

Normalization is a method in SQL to organize data in tables in such a way that:

- There is no **repetition of data**,
- Data is stored efficiently,
- There are no **update/delete anomalies**, and
- We can **avoid duplicate** and unnecessary data.



1NF

A table is in 1st Normal Form if:

- Each column contains only atomic (indivisible) values.
- There are no repeating groups or arrays.

Atomic means – Single value in one cell, not a list.

1F

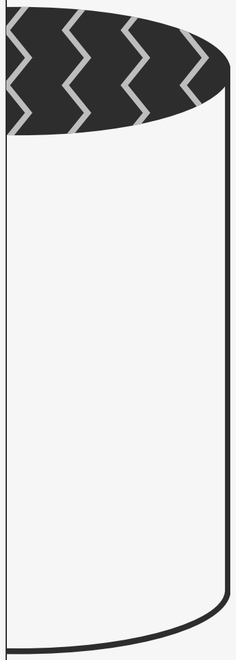
Student_ID	Name	Phone_Numbers
1	Raj	9876543210, 9123456789

Student_ID	Name	Phone1	Phone2
1	Raj	9876543210	9123456789

2NF (Second Normal Form)

A table is in 2NF if:

- *It is in 1NF, and*
- *No partial dependency exists (i.e., non-key columns depend on the whole primary key).*



Un Normalized Example

```
CREATE TABLE StudentCourseRaw (
    StudentID INT,
    CourseID INT,
    StudentName VARCHAR(50),
    CourseName VARCHAR(50)
);

INSERT INTO StudentCourseRaw VALUES
(1, 101, 'Priya', 'Physics'),
(1, 102, 'Priya', 'Chemistry');
```

2f-Making three tables instead of one

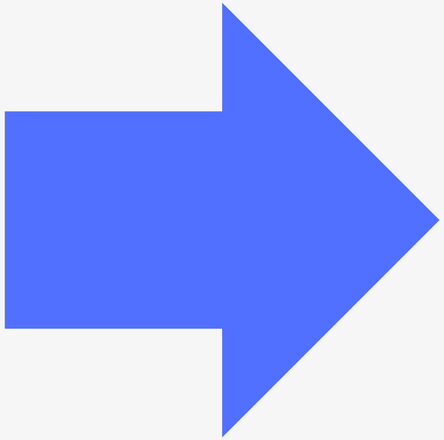
```
CREATE TABLE Student (
    StudentID INT PRIMARY
KEY,
    StudentName
VARCHAR(50)
);

INSERT INTO Student
VALUES
(1, 'Priya');
```

```
CREATE TABLE Course (
    CourseID INT PRIMARY
KEY,
    CourseName
VARCHAR(50)
);

INSERT INTO Course
VALUES
(101, 'Physics'),
(102, 'Chemistry');
```

```
CREATE TABLE StudentCourse (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID,
CourseID),
    FOREIGN KEY (StudentID)
REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID)
REFERENCES Course(CourseID)
);

INSERT INTO StudentCourse
VALUES
(1, 101),
(1, 102);
```

Final Result: Fully Normalized (2NF Achieved)

- StudentName depends only on StudentID.
- CourseName depends only on CourseID.
- Composite key table StudentCourse has no partial dependency.



Query to Reconstruct Original View

```
SELECT sc.StudentID, sc.CourseID,
s.StudentName, c.CourseName
FROM StudentCourse sc
JOIN Student s ON sc.StudentID =
s.StudentID
JOIN Course c ON sc.CourseID =
c.CourseID;
```

3F

- A table is in 3NF if:
- It is in 2NF, and
- There is no transitive dependency (i.e., non-key depends on another non-key).

```
CREATE TABLE AddressRaw (
    AddressID INT,
    Area VARCHAR(50),
    Pincode INT,
    City VARCHAR(50)
);

INSERT INTO AddressRaw VALUES
(1, 'Laxmi Nagar', 110092, 'Delhi'),
(2, 'Andheri', 400053, 'Mumbai');
```

Problem:
City depends on Pincode, not directly on AddressID.
So this is transitive dependency, and violates 3NF.

Convert to 3NF (Remove Transitive Dependency)

```
CREATE TABLE PincodeInfo (
    Pincode INT PRIMARY KEY,
    City VARCHAR(50)
);

INSERT INTO PincodeInfo VALUES
(110092, 'Delhi'),
(400053, 'Mumbai');
```

```
CREATE TABLE Address (
    AddressID INT PRIMARY KEY,
    Area VARCHAR(50),
    Pincode INT,
    FOREIGN KEY (Pincode)
REFERENCES PincodeInfo(Pincode)
);

INSERT INTO Address VALUES
(1, 'Laxmi Nagar', 110092),
(2, 'Andheri', 400053);
```

Final Result: Fully Normalized (3NF Achieved)

- City now depends only on Pincode, not on AddressID.
- Address table has no transitive dependency.
- This structure satisfies 3NF.

Query to Reconstruct Original View

```
SELECT a.AddressID,
a.Area, a.Pincode, p.City
FROM Address a
JOIN PincodeInfo p ON
a.Pincode = p.Pincode;
```

BCNF

- **table is in BCNF if:**
- **It is in 3NF, and**
- **Every determinant is a candidate key.**
- **In other words, if any non-primary attribute determines other attributes, it must also be a candidate key.**

- Each Mobile Number Prefix belongs to a specific SIM Company
- → e.g., 98 → Jio, 99 → Airtel
- But if we store:
- Full Mobile Number
- SIM Company
- Prefix
- Then Prefix → SIMCompany, but Prefix is not a key → violates BCNF.

```
CREATE TABLE SimDataRaw (
 MobileNumber VARCHAR(10),
 Prefix VARCHAR(2),
 SIMCompany VARCHAR(20)
 );

INSERT INTO SimDataRaw VALUES
 ('9812345678', '98', 'Jio'),
 ('9911122233', '99', 'Airtel'),
 ('9823456789', '98', 'Jio');
```

MobileNumber	Prefix	SIMCompany
9812345678	98	Jio
9911122233	99	Airtel
9823456789	98	Jio

Problem:

Primary Key = MobileNumber

But Prefix → SIMCompany (Prefix determines SIM company)

Prefix is not a candidate key, hence BCNF is violated

PrefixCompany — Tells which prefix belongs to which SIM company

```
CREATE TABLE PrefixCompany (
    Prefix VARCHAR(2) PRIMARY KEY,
    SIMCompany VARCHAR(20)
);

INSERT INTO PrefixCompany VALUES
('98', 'Jio'),
('99', 'Airtel');
```

MobileData — Stores full mobile numbers and their prefix

```
CREATE TABLE MobileData (
    MobileNumber VARCHAR(10) PRIMARY KEY,
    Prefix VARCHAR(2),
    FOREIGN KEY (Prefix) REFERENCES
PrefixCompany(Prefix)
);

INSERT INTO MobileData VALUES
('9812345678', '98'),
('9911122233', '99'),
('9823456789', '98');
```

Final: Fully Normalized (BCNF Achieved)

- No non-key determines another non-key.
- All determinants are candidate keys.

```
SELECT m.MobileNumber,
m.Prefix, p.SIMCompany
FROM MobileData m
JOIN PrefixCompany p ON
m.Prefix = p.Prefix;
```

MobileNumber	Prefix	SIMCompany
9812345678	98	Jio
9911122233	99	Airtel
9823456789	98	Jio