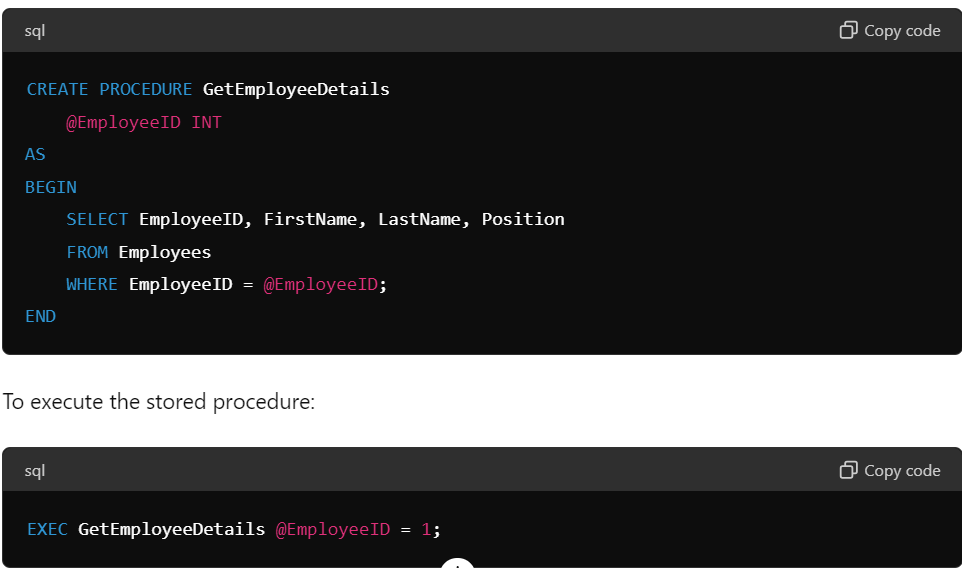
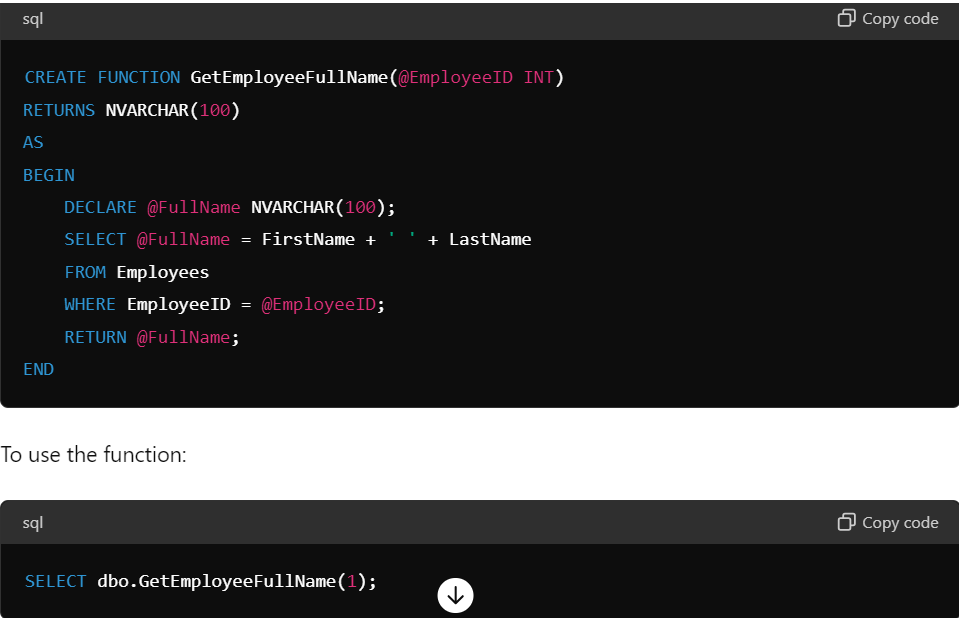
**What is a stored procedure, and how is it used?**

**Answer:** A stored procedure is a prepared SQL code that can be saved and reused. It allows for the encapsulation of complex queries and business logic in the database. Stored procedures can take input parameters, execute SQL statements, and return results or output parameters.



**What is a Function, and how is it used?**

**Answer:** A function in SQL is a set of instructions that performs a specific task and returns a value. You can use functions to simplify and reuse code. They are helpful for performing calculations, formatting data, and more. Functions can be used in various parts of SQL queries, such as in the SELECT list, WHERE clause, or JOIN conditions.



**Explain the concept of indexing and how it improves query performance.**

**Answer:** Indexing is a database optimization technique used to speed up the retrieval of rows from a table. An index creates an entry for each value that points to the row containing that value, making data retrieval much faster. Indexes are particularly useful for columns that are frequently searched or used in JOIN conditions.

**Example:**

CREATE INDEX idx\_employee\_lastname

ON Employees (LastName);

An index on the LastName column helps speed up queries that filter or sort by LastName.

**What is a composite index, and when would you use it?**

**Answer:** A composite index is an index on multiple columns of a table. It is used when queries often filter or sort on more than one column. Composite indexes can significantly improve the performance of such queries by providing a more efficient way to look up and retrieve the data.

**Example:**

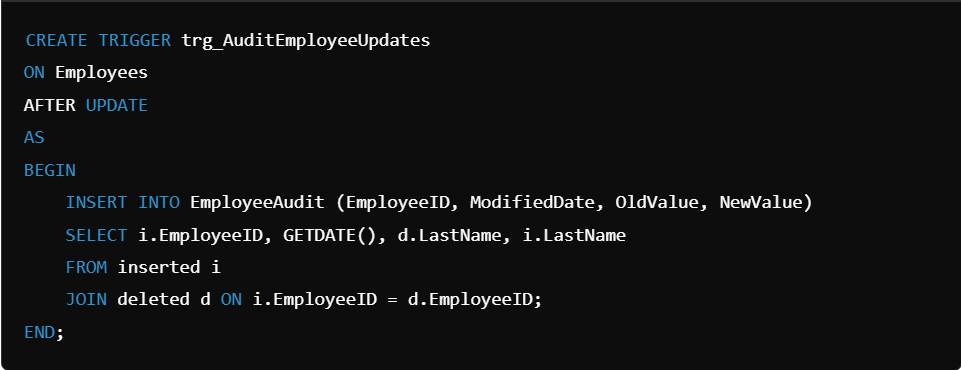
CREATE INDEX idx\_employee\_lastname\_firstname

ON Employees (LastName, FirstName);

This index is useful for queries that filter or sort by both LastName and FirstName.

**What are SQL triggers, and how do they work?**

**Answer:** SQL triggers are special types of stored procedures that automatically execute in response to certain events on a particular table or view. Triggers can be used to enforce business rules, validate data, or synchronize data across tables. They can be set to fire before or after an INSERT, UPDATE, or DELETE operation.



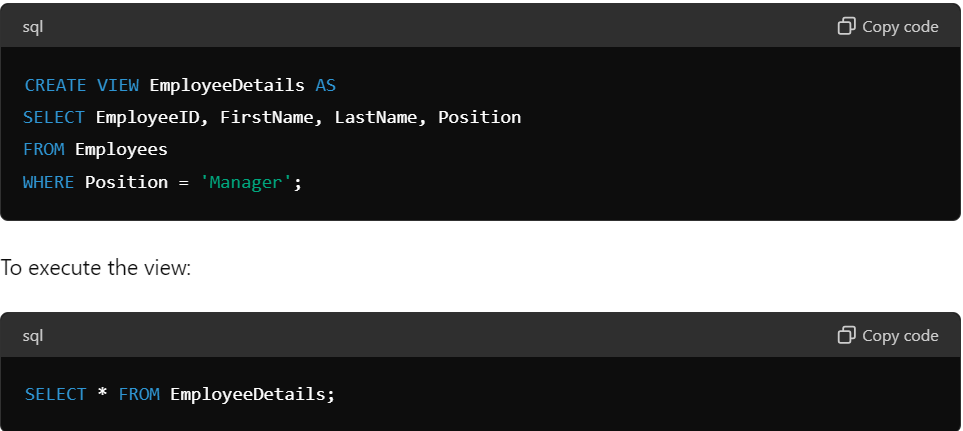
This trigger logs changes to the LastName column in the EmployeeAudit table.

There are six Triggers allowed to use in the MySQL database:

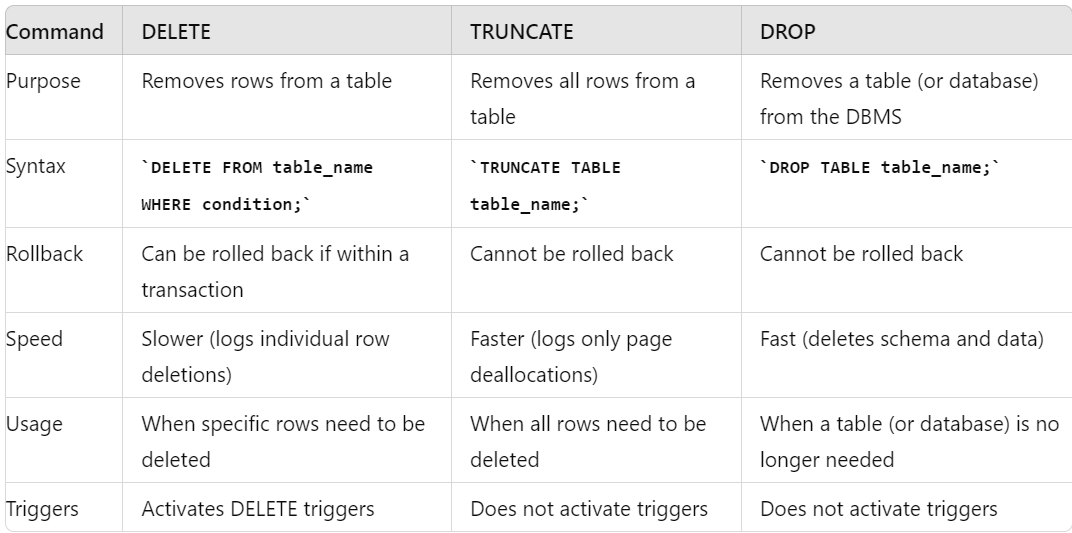
* Before Insert
* After Insert
* Before Update
* After Update
* Before Delete
* After Delete

**What is views, create and execute views in MySQL?**

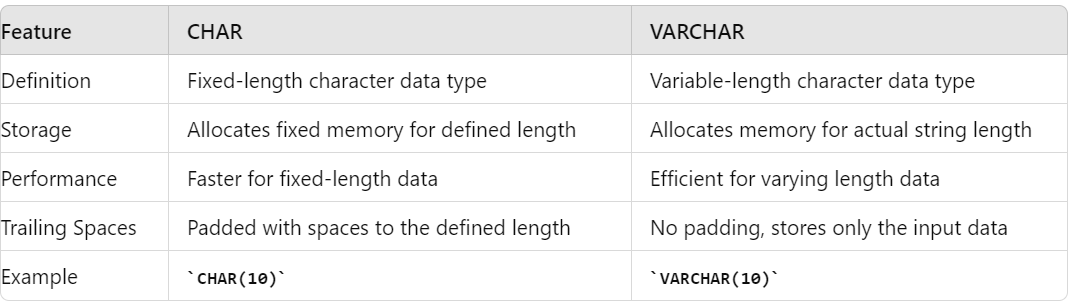
**Answer:** A view in MySQL is a virtual table that is based on the result set of an SQL query. Views are used to simplify complex queries, encapsulate logic, and provide a layer of security by restricting access to specific data.



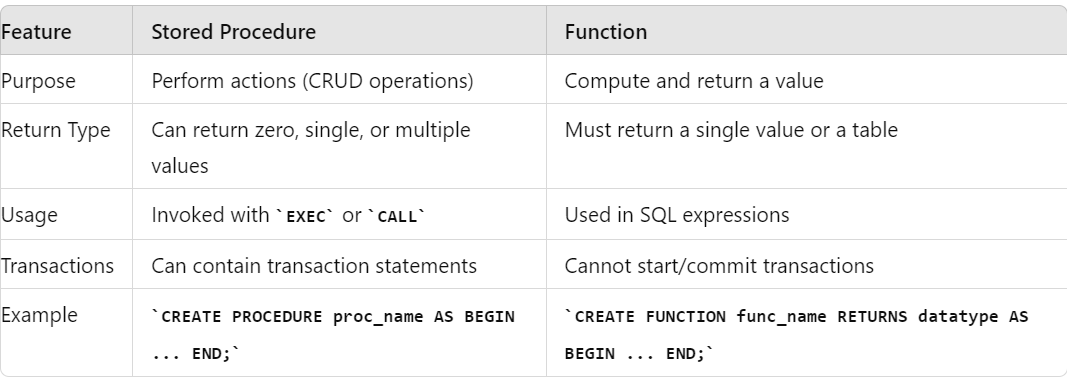
**Explain the difference between DELETE, TRUNCATE, and DROP commands in SQL**



**Difference between char and varchar.**

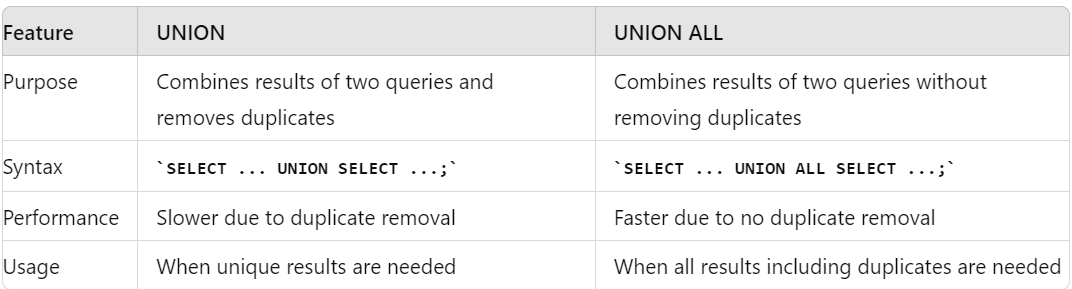


**Difference between a stored procedure and a function in SQL**

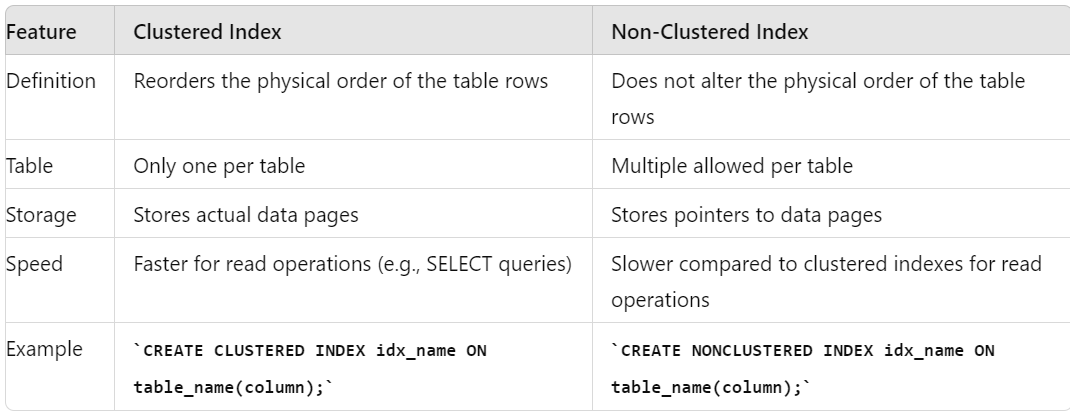
****



**What is the difference between UNION and UNION ALL in SQL**



**Difference between clustered and non-clustered index**



**How can you enhance the performance of a stored procedure?**

**Answer:**

1. **Indexing:** Ensure that the tables involved in the stored procedure have appropriate indexes.
2. **Avoid Cursors:** Use set-based operations instead of cursors wherever possible.
3. **Parameter Sniffing:** Optimize for parameter values to improve performance.
4. **Query Optimization:** Analyze and optimize the SQL queries within the stored procedure.
5. **Use EXISTS Instead of COUNT:** For checking the existence of rows, use EXISTS instead of COUNT.
6. \**Avoid SELECT :* Specify only the columns you need.
7. **Avoid Unnecessary Calculations and Conversions:** Reduce the use of functions on indexed columns.
8. **Update Statistics:** Ensure that the database statistics are up to date.
9. **Execution Plans:** Review and optimize the execution plans for queries.
10. **Partitioning:** Use table partitioning for large tables.

**How can you duplicate a table structure and data to another table in a single query?**

**Answer:** You can use the SELECT INTO statement to duplicate a table structure along with its data.

SELECT \* INTO NewTable FROM OriginalTable;

This will create a new table NewTable with the same structure and data as OriginalTable.

**How can you write a SQL query to find the top 10 records of something?**

**Answer:** You can use the TOP clause in SQL Server or LIMIT clause in MySQL to find the top 10 records. **SQL Server Example:**

SELECT TOP 10 \* FROM Employees ORDER BY Salary DESC;

**MySQL Example:**

SELECT \* FROM Employees ORDER BY Salary DESC LIMIT 10;

**What database are you using in your project and what is its version?**

**Answer:** This is a personalized question. You would provide the name of the database and its version based on your current project. For example: "We are using Microsoft SQL Server 2019 in our project."

### **Normalization in SQL: Definitions and Examples**

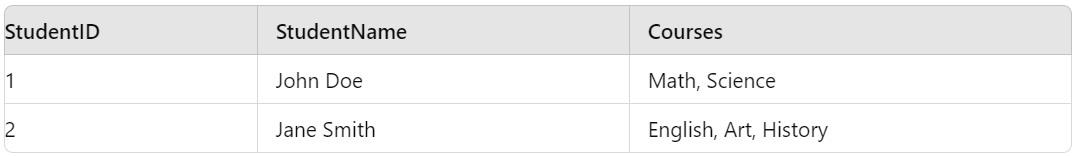
Normalization is a process in database design that organizes tables to reduce redundancy and improve data integrity. It involves dividing a database into two or more tables and defining relationships between them. The main goal is to eliminate redundant data and ensure data dependencies make sense to avoid any anomalies during data operations.

#### First Normal Form (1NF)

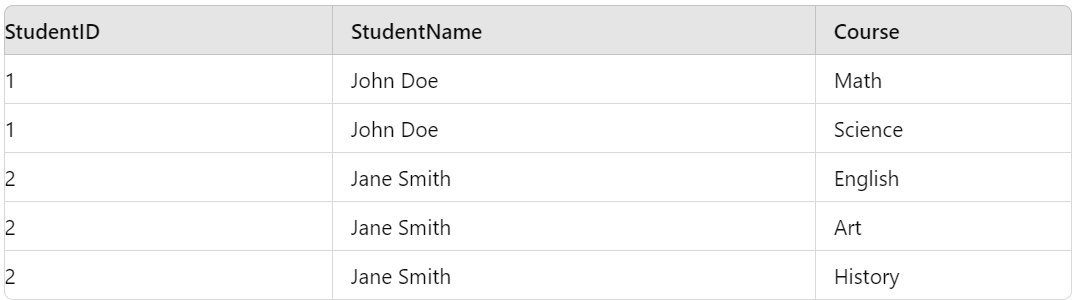
**Definition:** A table is in the First Normal Form if:

* It contains only atomic (indivisible) values.
* Each column contains values of a single type.
* Each column contains unique values (or only values relevant to that column).

**Example:** Consider the following table that is not in 1NF:



To convert this to 1NF, we need to ensure each column has only atomic values:

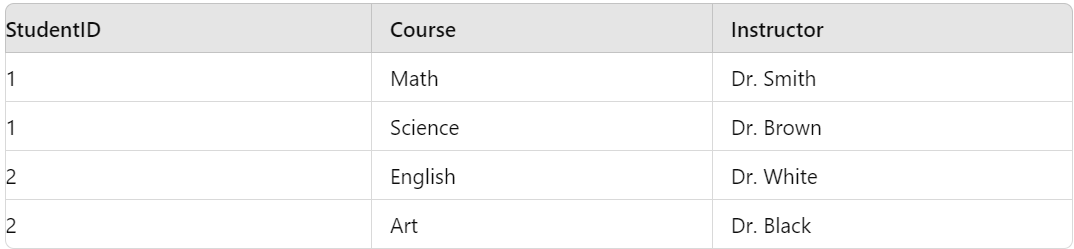


#### Second Normal Form (2NF)

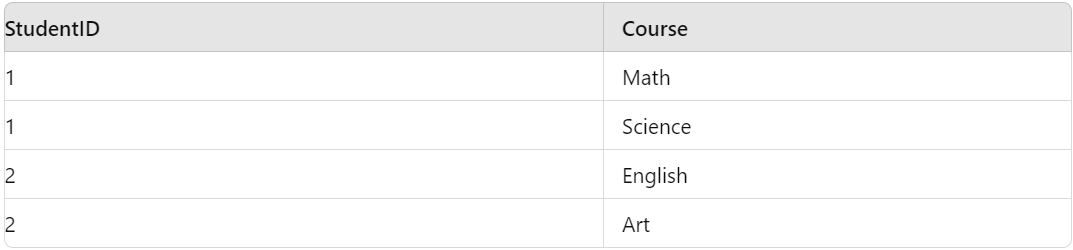
**Definition:** A table is in the Second Normal Form if:

* It is in 1NF.
* All non-key attributes are fully functional dependent on the primary key. This means there should be no partial dependency of any column on the primary key.

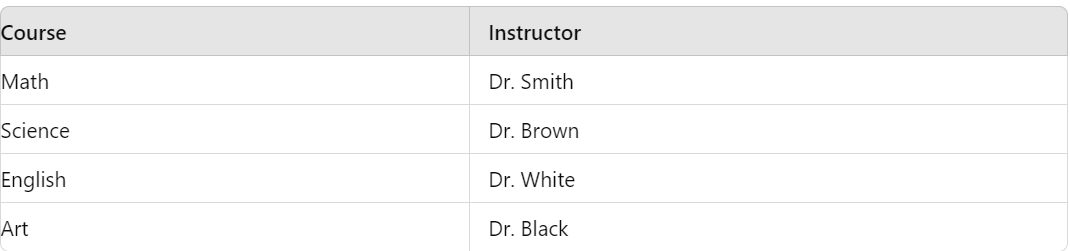
**Example:** Consider the following table that is in 1NF but not in 2NF:



To convert this to 2NF, we need to remove the partial dependency by creating two tables: **Students Table:**



**Instructors Table:**

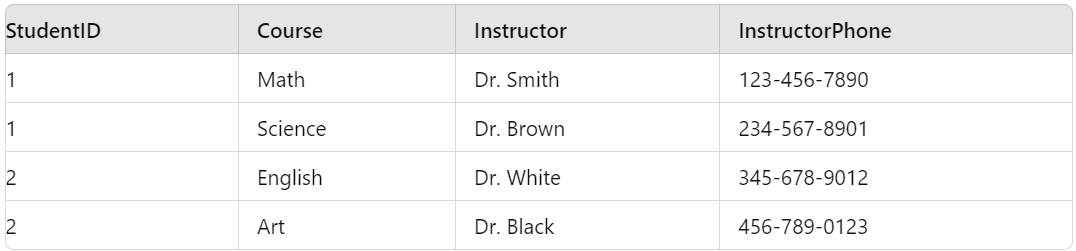


#### Third Normal Form (3NF)

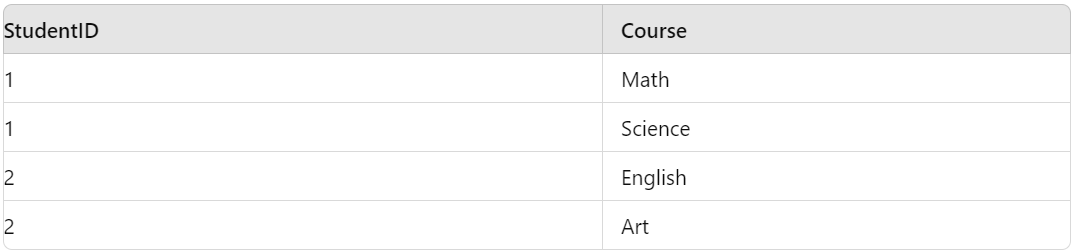
**Definition:** A table is in the Third Normal Form if:

* It is in 2NF.
* All non-key columns (attributes) depend only on the primary key, and not on any other non-key columns.

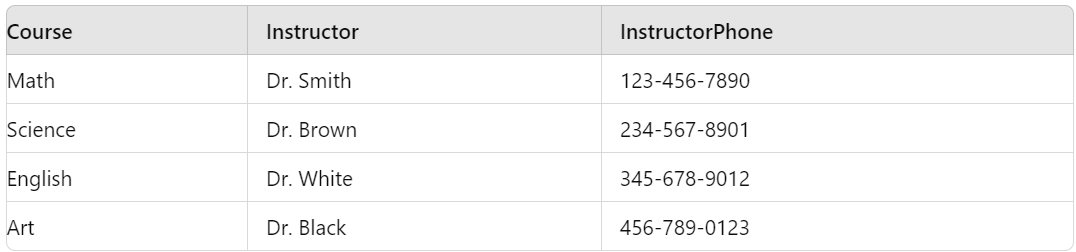
**Example:** Consider the following table that is in 2NF but not in 3NF:



To convert this to 3NF, we need to remove the transitive dependency: **Students Table:**



**Instructors Table:**

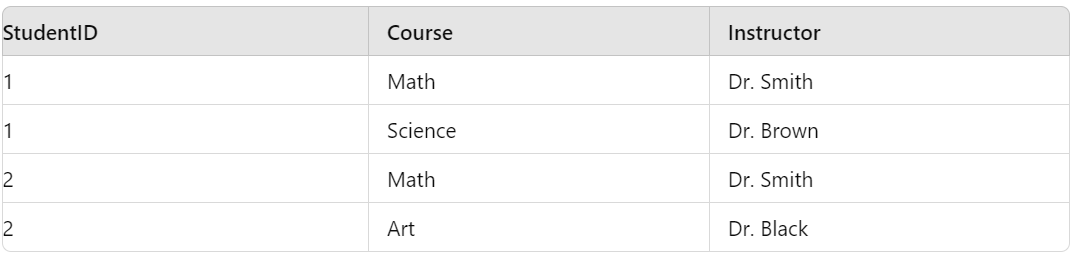


#### Boyce-Codd Normal Form (BCNF)

**Definition:** A table is in Boyce-Codd Normal Form if:

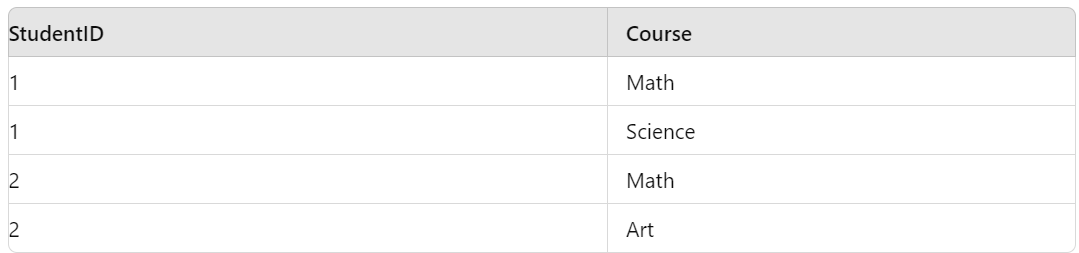
* It is in 3NF.
* For every functional dependency (X → Y), X should be a super key.

**Example:** Consider the following table that is in 3NF but not in BCNF:

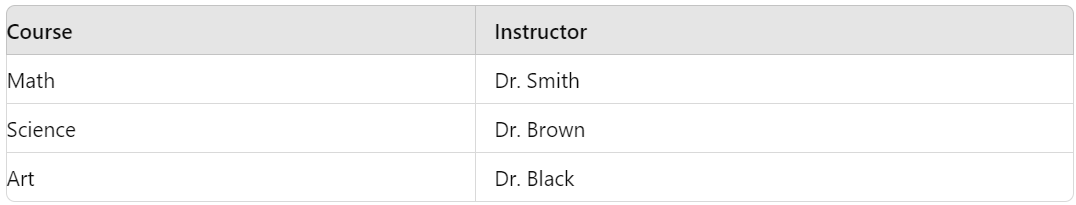


Here, (Course → Instructor) violates BCNF because Course is not a super key. To convert this to BCNF:

**Students Table:**



**Courses Table:**



This ensures that each non-trivial functional dependency has a super key as its determinant.

Summary

Normalization helps in organizing data efficiently by reducing redundancy and ensuring data integrity through various normal forms. Each successive normal form addresses specific types of anomalies and dependencies, leading to a well-structured database design.

### **Schema Design for Healthcare Sector: Doctors and Patients**

#### Tables and Relationships

1. **Doctors**
2. **Patients**
3. **Appointments**
4. **Specializations**

#### Tables and Their Attributes:

**Doctors:**

* doctor\_id INT (Primary Key)
* first\_name VARCHAR(50)
* last\_name VARCHAR(50)
* specialization\_id INT (Foreign Key)
* phone\_number VARCHAR(15)
* email VARCHAR(100)

**Specializations:**

* specialization\_id INT (Primary Key)
* specialization\_name VARCHAR(100)

**Patients:**

* patient\_id INT (Primary Key)
* first\_name VARCHAR(50)
* last\_name VARCHAR(50)
* date\_of\_birth DATE
* gender CHAR(1)
* phone\_number VARCHAR(15)
* email VARCHAR(100)

**Appointments:**

* appointment\_id INT (Primary Key)
* doctor\_id INT (Foreign Key)
* patient\_id INT (Foreign Key)
* appointment\_date DATETIME
* reason TEXT

#### Relationships:

* **Doctors:** doctor\_id is the primary key.
* **Specializations:** specialization\_id is the primary key.
* **Patients:** patient\_id is the primary key.
* **Appointments:** appointment\_id is the primary key.
* **Foreign Keys:**
  + Doctors(specialization\_id) REFERENCES Specializations(specialization\_id)
  + Appointments(doctor\_id) REFERENCES Doctors(doctor\_id)
  + Appointments(patient\_id) REFERENCES Patients(patient\_id)

### **Example Queries**

#### Find all appointments for a specific doctor

sql

Copy code

SELECT a.appointment\_id, p.first\_name AS patient\_first\_name, p.last\_name AS patient\_last\_name, a.appointment\_date, a.reason

FROM Appointments a

JOIN Patients p ON a.patient\_id = p.patient\_id

JOIN Doctors d ON a.doctor\_id = d.doctor\_id

WHERE d.first\_name = 'John' AND d.last\_name = 'Doe';

#### List all patients under a specific doctor

sql

Copy code

SELECT p.first\_name, p.last\_name, p.date\_of\_birth, p.gender, p.phone\_number, p.email

FROM Patients p

JOIN Appointments a ON p.patient\_id = a.patient\_id

JOIN Doctors d ON a.doctor\_id = d.doctor\_id

WHERE d.first\_name = 'John' AND d.last\_name = 'Doe';

#### Get all doctors with their specializations

sql

Copy code

SELECT d.first\_name, d.last\_name, s.specialization\_name, d.phone\_number, d.email

FROM Doctors d

JOIN Specializations s ON d.specialization\_id = s.specialization\_id;

This schema design allows efficient querying and management of healthcare-related data, such as tracking doctor-patient appointments and managing doctor specializations.