Normalization

Normalization is a way of organizing data in a database to make it more efficient and reduce redundancy.
Divides data into separate tables (like categories).

1NF

To be in **1NF**, the table must have:

- Atomic (indivisible) values.
- No repeating groups.

?)Convert the following table into 1NF: - Table: Orders - Columns: OrderID, CustomerName, ProductNames (where ProductNames might contain multiple product names separated by commas).

create table orders (order_id int, customer_name varchar(50), product_name varchar(50));

insert into orders (order_id, customer_name, product_name) values (1, 'Alice', 'Coffee'), (1, 'Alice', 'Tea'), (2, 'Bob', 'Cappuccino'), (2, 'Bob', 'Latte'), (3, 'Charlie', 'Espresso');

2NF

Must be in 1NF: The table should have atomic values, with no repeating groups.

- **No Partial Dependencies**: Every non-key attribute should depend on the **whole primary key**, not just part of it.
- This means if there's a composite primary key (a primary key with multiple columns), each non-key attribute should depend on all parts of that key

?) Normalize the following table to 2NF: - Table: Student_Courses - Columns: StudentID, StudentName, CourseID, CourseName, InstructorID, InstructorName make query

To normalize the Student_Courses table into **2NF**, we must ensure that the table is in **1NF** and that all non-key attributes are fully functionally dependent on the primary key. In this case, StudentID and CourseID together form a composite primary key in the Student_Courses table.

create table students (student_id int primary key, student_name varchar(50) not null);
insert into students (student_id, student_name) values (1, 'Alice'), (2, 'Bob');

create table courses (course_id int primary key, course_name varchar(50) not null, instructor_id int, instructor_name varchar(50));
insert into courses (course_id, course_name, instructor_id, instructor_name) values (101, 'Math', 1001, 'Dr. Smith'), (102, 'Science', 1002, 'Dr. Jones');

create table student_courses (student_id int, course_id int, primary key (student_id, course_id), foreign key (student_id) references students(student_id), foreign key (course_id) references courses(course_id));
insert into student_courses (student_id, course_id) values (1, 101), (2, 102), (1, 102);

3NF

Must be in 2NF: It should already satisfy the rules of 2NF.

- **No Transitive Dependencies**: Non-key attributes should not depend on other non-key attributes, only on the primary key.
- In other words, a non-key attribute shouldn’t indirectly depend on the primary key through another non-key attribute.

?) Normalize the following table to 3NF: - Table: Sales - Columns: S, ProductID, ProductName, CategoryID, CategoryName, SaleDate make query

To normalize the Sales table into **3 NF**, we must first ensure it is in **2N F**, and then remove any transitive dependencies. In this case, SalesID could be the primary key, but attributes such as ProductName and CategoryName depend only on ProductID and CategoryID, respectively. This structure suggests transitive dependencies, as ProductName and CategoryName are not directly dependent on SalesID.

create table products (product_id int primary key, product_name varchar(50) not null, category_id int, foreign key (category_id) references categories(category_id));
insert into products (product_id, product_name, category_id) values (101, 'Coffee', 10), (102, 'Tea', 10), (103, 'T-shirt', 20);

create table categories (category_id int primary key, category_name varchar(50) not null);
insert into categories (category_id, category_name) values (10, 'Beverages'), (20, 'Apparel');

create table sales (sales_id int primary key, product_id int, sale_date date not null, foreign key (product_id) references products(product_id));
insert into sales (sales_id, product_id, sale_date) values (1, 101, '2024-01-01'), (2, 102, '2024-01-02'), (3, 103, '2024-01-03');