Makara Ramoabi

20240045

Database

1.**Scenario analysis**

Normalization transforms a hospital database from redundant chaos to efficient structure. Start with unnormalized: PatientID, Name, DOB, Phone, ApptID, ApptDate, DoctorID, DoctorName, Dept. Issues: duplicate names/phones across appointments cause update anomalies.

To 3NF:

Patients Table (PK: PatientID)

PatientID	Name	DOB	Phone
1	Alice	1990-01-01	123-456

Appointments Table (PK: ApptID; FK: PatientID, DoctorID)

ApptID	PatientID	DoctorID	ApptDate
101	1	D1	2026-02-01

Separate Doctors (PK: DoctorID, Name, Dept). Integrity ensured: referential integrity via FKs prevents orphan appointments; no transitive deps (e.g., Dept depends on DoctorID). Updates atomic—one change propagates. Queries join for views, reducing anomalies 90%.

2.Normalization Task

Unnormalized Students: StudentID, Name, CourseID, CourseName, DeptName, Marks.

FDs: StudentID → Name; CourseID → CourseName, DeptName; {StudentID, CourseID} → Marks.

To 1NF: Atomic values, repeat rows removed.

2NF: Split partial deps—Students(StudentID, Name); Courses(CourseID, CourseName, DeptName).

Issue: CourseID → DeptName (partial on composite key if any).

3NF: Transitive CourseID → DeptName → ? No, but split: Departments(DeptID, DeptName); Courses(CourseID, DeptID).

Final: Enrollments(StudentID, CourseID, Marks) with FKs.

Eliminates redundancy—update DeptName once; anomalies gone.

3.Design Goals

Normalization boosts e-commerce performance by slashing redundancy—store customer details once, reference via FKs in Orders, cutting storage 50-70%. Joins enable fast analytics: sum sales per product without dupes. Indexes on PK/FKs speed lookups (e.g., user cart queries). Less data transfer reduces I/O latency; queries optimize via query planner. In high-traffic sites, avoids bloat slowing backups/restores. Trade-off: more joins, but indexing/CDNs mitigate. Denormalize views for reads (e.g., product+reviews). Overall, scales to millions of transactions, ensuring integrity amid peaks.

4.2NF and 3NF Summary

2NF: 1NF + no partial dependencies—non-prime attrs fully depend on entire candidate key. Example: Orders(OrderID, ProductID, Qty, ProductName)—split ProductName to Products as it depends only on ProductID.

3NF: 2NF + no transitive dependencies—non-prime attrs depend directly on key, not other non-primes. Example: Employees(EmpID, DeptID, DeptName)—DeptName depends on DeptID; split to Departments(DeptID, DeptName).