Denormalization in Databases

Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean not doing normalization. It is an optimization technique that is applied after doing normalization.

In a traditional normalized database, we store data in separate logical tables and attempt to minimize redundant data. We may strive to have only one copy of each piece of data in database.

For example, in a normalized database, we might have a Courses table and a Teachers table. Each entry in Courses would store the teacherID for a Course but not the teacherName. When we need to retrieve a list of all Courses with the Teacher's name, we would do a join between these two tables.

In some ways, this is great; if a teacher changes his or her name, we only have to update the name in one place.
The drawback is that if tables are large, we may spend an unnecessarily long time doing joins on tables.
Denormalization, then, strikes a different compromise. Under denormalization, we decide that we're okay with some redundancy and some extra effort to update the database in order to get the efficiency advantages of fewer joins.

Pros of Denormalization:-

1. Retrieving data is faster since we do fewer joins
2. Queries to retrieve can be simpler(and therefore less likely to have bugs), since we need to look at fewer tables.

Cons of Denormalization:-

1. Updates and inserts are more expensive.
2. Denormalization can make update and insert code harder to write.
3. Data may be inconsistent . Which is the "correct" value for a piece of data?
4. Data redundancy necessitates more storage.

In a system that demands scalability, like that of any major tech company, we almost always use elements of both normalized and denormalized databases.