Optimistic locking: its basic idea is that every time a transaction update is submitted, we want to see if the thing to be modified has been modified by other transactions since the last read. If it is modified, the update will fail.
EX we have transaction A access a record, optimistic locking means that record is not added lock on it so other transactions can access it as well. To solve the conflict, we need to add a version in the field. If A accesses the record and that version is 1 and you commit it and check what the version is now. If we get a different version, It shows an error.

In the entire data processing process, the data is locked to ensure the maximum exclusivity of the operation. Of course, it adds a lot of overhead to the database and reduces performance. Generally, rely on the lock mechanism provided by the database.

How to solve the dead lock

1. Dead lock detect: innodb_deadlock_detect. Every time you add lock on the row it will judge whether it will cause the dead lock.
2. Using RAG, it's possible to predict the occurrence of deadlock in an OS.

   The resource allocation graph is the pictorial view of all allocated resources, available resources, and OS's current state. We can figure out how many resources are allocated to each process and how many resources will be needed in the future. Thus, we can easily avoid the deadlock. As every graph has vertices and edges, in the same way, RAG also has vertices and edges. RAG has two vertices: process vertex and resource vertex, and two edges: assignment edge and request edge. Mostly, we represent vertices with a rectangular shape and edges with a circular shape:

Sage:

The Saga pattern provides transaction management using a sequence of *local transactions*. A local transaction is the atomic work effort performed by a saga participant. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. If a local transaction fails, the saga executes a series of *compensating transactions* that undo the changes that were made by the preceding local transactions.