Database Week 7

Khaula Molapo

1.

For an e-commerce platform, a parallel database setup means data is divided across different servers that work together. For example, customer info, products, and orders are stored on separate nodes. When someone searches or buys something, the system sends the request to multiple servers at once, each one handling a part of the job. The results are combined and sent back quickly. Load balancing helps share the workload, and if one server fails, another takes over using backup data. This setup keeps the system running smoothly even when thousands of people are online. It improves speed, reliability, and makes it easy to scale up when traffic increases. During big sales or holidays, everything runs faster and doesn't crash. Basically, it's a group of databases working together to make one powerful and reliable system.

2.

A shared-nothing architecture means every server or node in the system works independently. Each has its own CPU, memory, and storage, and they don't share resources. Data is split between them, and each node handles its own part of the work. This design is great for performance and scaling because you can just add more nodes when the system grows. There's no fighting over memory or storage. The only downside is that it can get complicated when data from different nodes needs to be combined. Still, it's one of the best setups for large systems that need speed and flexibility.

3.

In PostgreSQL, you can test a parallel query by turning on the setting that allows multiple workers to process data at once. Then you run a large query, like selecting data from a big table with a filter. When the query runs, PostgreSQL splits the task so that different parts of the table are scanned at the same time. This makes the process faster, especially with huge datasets. However, small queries or simple searches might not benefit from it. Parallel

queries are best for heavy workloads where dividing the job across workers really improves speed and efficiency.

4.

If I were designing a parallel database, I'd split the data across different servers based on regions or categories. Each server handles its own data and processes queries in parallel with others. A main coordinator collects all the results and sends back the full answer to the user. This kind of setup helps the system handle large numbers of users without slowing down. It's faster, more reliable, and can be expanded easily by adding more servers. The main goal is better performance, less risk of failure, and a smoother experience for users even during heavy traffic.