Redis Not Suitable Use Cases Compared to GridGain

1. Distributed ACID Transactions - Redis lacks cross-node ACID support; GridGain supports 2PC.
2. Complex SQL Analytics - Redis lacks full SQL; GridGain supports ANSI SQL, joins, aggregates.
3. Co-located Compute - Redis doesn't support in-memory compute; GridGain enables compute/data colocation.
4. Trade Lifecycle - Redis can't manage consistent multi-entity updates; GridGain supports full ledger ops.
5. Real-Time Settlement - Redis Streams limited; GridGain supports full streaming pipelines.
6. Risk Aggregation - Redis can't aggregate across keys with SQL; GridGain supports MPP SQL.
7. Persistence - Redis requires reloads; GridGain has native durable memory.
8. Integration - Redis requires ETL for external data; GridGain acts as an integration hub.
9. Compliance Auditing - Redis lacks snapshots; GridGain includes full snapshot & recovery.
10. Structured Data - Redis struggles with complex schema; GridGain supports SQL schema & relationships.

Redis is Better Fit than GridGain

1. Ultra-Low Latency Caching - Redis is faster for key/value and session data.
2. Session Storage - Redis is ideal for JWT/session tokens.
3. Microservices - Redis is lightweight and perfect for config/token caches.
4. Leaderboards - Redis supports atomic counters and sorted sets.
5. Pub/Sub - Redis Streams/PubSub is simple and low-latency.
6. Feature Stores - Redis serves vector/JSON features in <1ms.
7. Temporary Caches - Redis is perfect for caching SQL/API queries.
8. Time-Series - RedisTimeSeries supports rollups/retention easily.
9. API Throttling - Built-in support for rate-limiting algorithms.
10. SaaS Tenant Isolation - Simple multitenancy with logical DBs.

Summary: When to Use Redis Over GridGain

Use Redis when:
- You need ultra-low latency caching or session storage.
- Your workload is read-heavy and stateless.
- You want rapid deployment and lower ops cost.
- You don’t need distributed SQL or transactional consistency.
- You want vector, time-series, or search modules.

Redis is best for:
- Token/session storage
- Leaderboards
- API rate limiting
- Simple pub/sub messaging
- Microservices configs