inPlaceStatusService , the key use to save in redis is"place:" + quoteId + ":" + placeId; in reporitory will findbyquote id work? quote id is not the key right

Your understanding is correct — if Redis is storing entries by using keys in the pattern `"place:" + quoteId + ":" + placeId`, then the **quoteId itself is not the Redis key**, but rather just a segment of it. Because of this, `findByQuoteId()` **will not automatically work** when using `ReactiveCrudRepository` with Redis, since the repository methods map directly to keys rather than nested object fields.

Why `findByQuoteId()` won't work directly

Redis does not natively support querying by fields embedded inside values or by substrings of keys. The `ReactiveCrudRepository` on Redis is tied to direct key-value lookups. In your case:

- The key stored in Redis looks like `place:Q123:P045`
- The value is a `PlaceStatusRecord` object carrying `quoteId`, `placeId`, etc.
- Redis indexing only maps to the *entire key string*, not just `quoteId`.

Hence, `repository.findByQuoteId("Q123")` won't match any results unless an additional index or structured lookup mechanism exists.

Three correct approaches

1. **Programmatic filter after reading all entries**
   While not optimal at scale:

```
redis.opsForScan().scan()
    .filter(key -> key.startsWith("place:" + quoteId + ":"))
    .flatMap(key -> redis.opsForValue().get(key))
    .collectList();
```

2. **Maintain a secondary set per quote**
   Whenever saving to Redis:

```
redis.opsForValue().set(key, record, ttl);
redis.opsForSet().add("places-by-quote:" + quoteId, key);
```

   Then retrieval by quoteId becomes:

```
redis.opsForSet().members("places-by-quote:" + quoteId)
    .flatMap(key -> redis.opsForValue().get(key))
    .collectList();
```

3. **Switch repository to in-memory abstraction**
   If you use a simple `ConcurrentMap` or relational store as backing for repository queries, you can preserve full `findByQuoteId()` semantics while Redis keeps only expirable keys.

Recommended change

For reactive Redis setups like in your uploaded project, **option (2)** (maintaining a secondary lookup set) is the most scalable and idiomatic solution. It allows TTL-driven behavior while providing efficient filtered retrieval per `quoteId`, matching well with your **Kafka + SSE event pipeline** structure.

⁂

1. spring-reactive-redis-ttl-kafka-deads-letter.pdf