Assignment 4 problems：

Part 1 Database design

**Why choose dynamodb:**

## Scalability

* Seamless scaling: DynamoDB automatically scales to handle traffic spikes during peak ski season without manual intervention
* No capacity planning: Can handle millions of lift ride records without database restructuring
* Partition-based: Distributes ski resort data across many partitions for parallel processing

## Performance

* Single-digit millisecond responses: Critical for real-time lift ride recording
* Consistent performance: Maintains speed regardless of data size (whether tracking 100 or 100,000 skiers)
* Global tables: Can replicate data across regions for multi-resort operations

## Schema Flexibility

* Schema-less design: Easy to add new attributes (like equipment rentals or lesson data) without migrations
* Varied data types: Accommodates different data models for different ski resort operations

## Operational Benefits

* Fully managed: No database administration overhead
* High availability: 99.99% availability SLA with multi-AZ replication
* Pay-per-use pricing: Cost effective for seasonal ski resort traffic patterns
* Automated backups: Built-in point-in-time recovery

## Use Case Alignment

* Time-series data: Ideal for lift ride tracking which is time-series in nature
* Key-based access patterns: Your queries match DynamoDB's access pattern strengths
* Sparse data: Only store attributes that exist (not all skiers use all lifts)

## Compared to Alternatives:

vs. RDS:

* No need for complex joins (your queries are key-based)
* Better handles write-heavy workloads (constant lift ride recording)
* No schema rigidity when business requirements change

vs. Redis:

* Better persistence guarantees (Redis is primarily in-memory)
* Superior horizontal scaling for large datasets
* More sophisticated querying with GSI and LSI options
* Built-in backup and disaster recovery

Cons:

Must design the schema with target query in mind. RDS is more flexible with different queries.

**Why skierId as partition key and combination of dayId#liftId#timestamp as sort key:**

Most of the query are based on skierId.

Concatenation of the attributes gives more freedom on making queries using “begin with” “end with”

**Why choosing resort-day and skier-day as GSI:**

The query is asking for unique skiers for resort/season/day. There are probably more resorts than seasons and days. Making season/day to be GSI partition key may make the partition too “hot”. why day is sort key than season -> same reason.

why we use skier-day GSI rather than make it LSI or use query “begin with”?:

### Global Secondary Index (current approach)

* Pros for fast reads:
  + Direct lookup using exact equality on both skierId and dayId
  + No need for begins\_with operations which can be less efficient
  + Smaller index size due to projected attributes
  + Separate capacity from base table (can provision higher read capacity)
* Cons:
  + Eventually consistent by default (can request strongly consistent reads at higher cost)
  + Additional write capacity needed to maintain the index

### Local Secondary Index (alternative)

* Pros for fast reads:
  + Strongly consistent reads available
  + Can be slightly faster than GSIs due to being co-located with base table items
* Cons:
  + Must be created at table creation time
  + Shares provisioned throughput with the base table
  + Still has the 10GB limit per partition

### Primary Table with begins\_with (no additional index)

* Pros:
  + Strongly consistent reads
  + No additional cost or complexity
* Cons:
  + Less efficient query pattern (range scan with begins\_with rather than equality)
  + Must retrieve all attributes (unless using projection expressions)