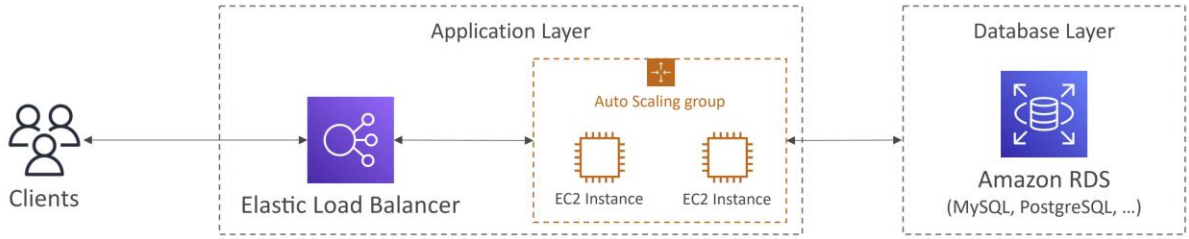
DynamoDB

Traditional Architecture



- Traditional applications leverage RDBMS databases
- These databases have the SQL query language
- Strong requirements about how the data should be modeled
- Ability to do query joins, aggregations, complex computations
- Vertical scaling (getting a more powerful CPU / RAM / IO)
- Horizontal scaling (increasing reading capability by adding EC2 / RDS Read Replicas)

NoSQL databases

- NoSQL databases are non-relational databases and are **distributed**
- NoSQL databases include MongoDB, DynamoDB, …
- NoSQL databases do not support query joins (or just limited support)
- All the data that is needed for a query is present in one row
- NoSQL databases don't perform aggregations such as "SUM", "AVG", …
- **NoSQL databases scale horizontally**

Amazon DynamoDB

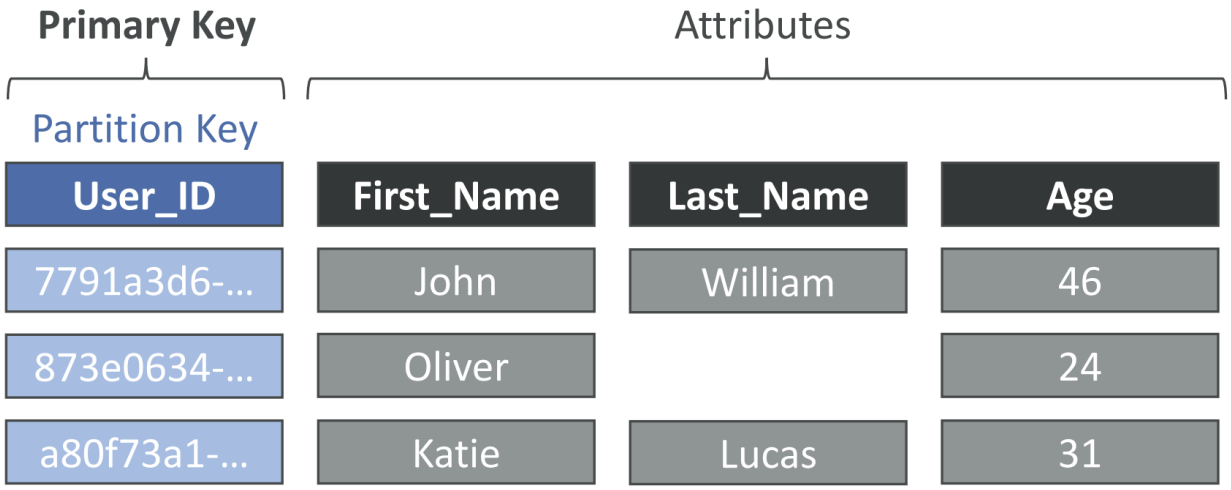
- Fully managed, highly available with replication across multiple AZs
- NoSQL database - not a relational database
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (low latency on retrieval)
- Integrated with IAM for security, authorization and administration
- Enables event driven programming with DynamoDB Streams
- Low cost and auto-scaling capabilities
- Standard & Infrequent Access (IA) Table Class

DynamoDB – Basics

- DynamoDB is made of **Tables**
- Each table has a **Primary Key** (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has **attributes** (can be added over time – can be null)
- Maximum size of an item is **400KB**
- Data types supported are:
  - **Scalar Types** – String, Number, Binary, Boolean, Null
  - **Document Types** – List, Map
  - **Set Types** – String Set, Number Set, Binary Set

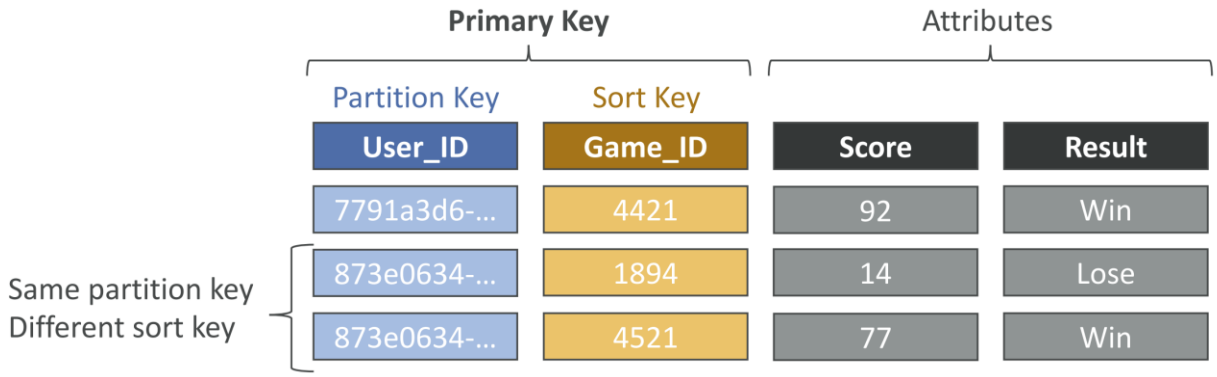
DynamoDB – Primary Keys

- Option 1: Partition Key (HASH)
  - Partition key must be unique for each item
  - Partition key must be "diverse" so that the data is distributed
  - Example: "User_ID" for a users table



DynamoDB – Primary Keys

- Option 2: Partition Key + Sort Key (HASH + RANGE)
  - The combination must be unique for each item
  - Data is grouped by partition key
  - Example: users-games table, "User_ID" for Partition Key and "Game_ID" for Sort Key



DynamoDB – Partition Keys (Exercise)

- We're building a movie database
- What is the best Partition Key to maximize data distribution?
  - movie_id
  - producer_name
  - leader_actor_name
  - movie_language

- "movie_id" has the highest cardinality so it's a good candidate
- "movie_language" doesn't take many values and may be skewed towards English so it's not a great choice for the Partition Key