DynamoDB Design Patterns

Intro to Dynamo

Dynamo is a **managed**, **scalable**, **NoSQL key-value**, **wide-column** database

managed: Dynamo exposes APIs and handles the rest

scalable: can handle up to 40k RPS; scales up and down with demand

NoSQL: not a relational database

key-value: think distributed hash table

wide-column: names / formats of item attributes varies from row-to-row

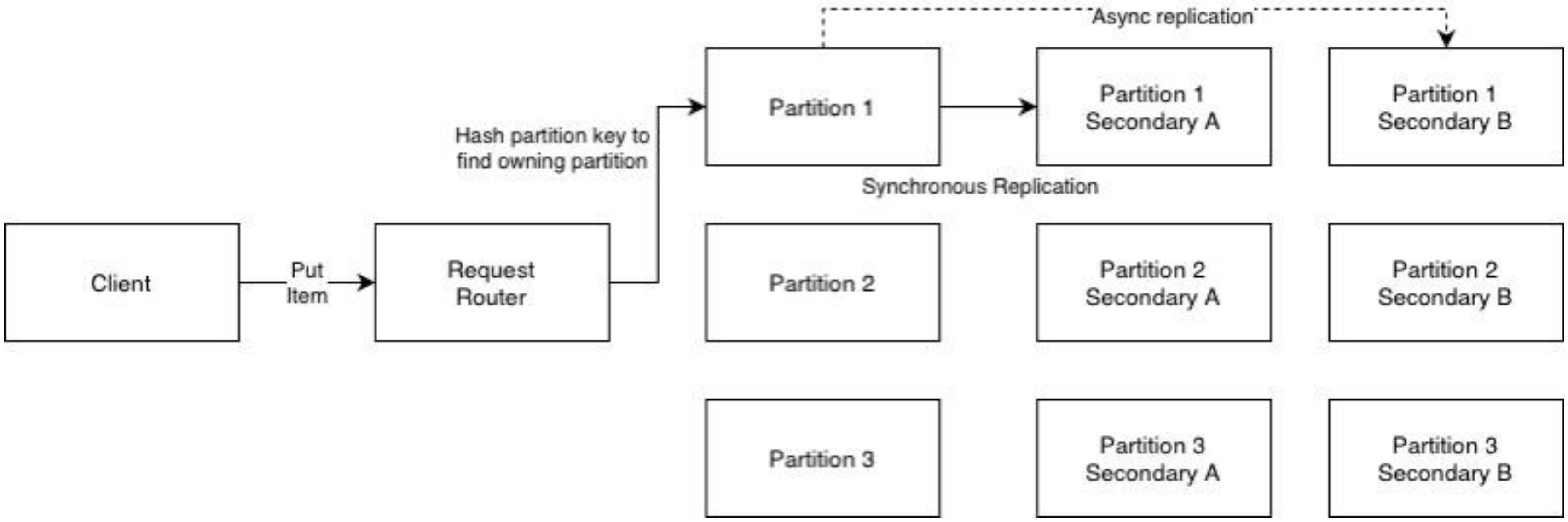
Why Dynamo?

- Highly available and scalable
- Strict query patterns guard against operations that won’t scale
- Pay only for the capacity you need
- Tightly integrated with AWS (IAM, CloudWatch, CloudFormation)

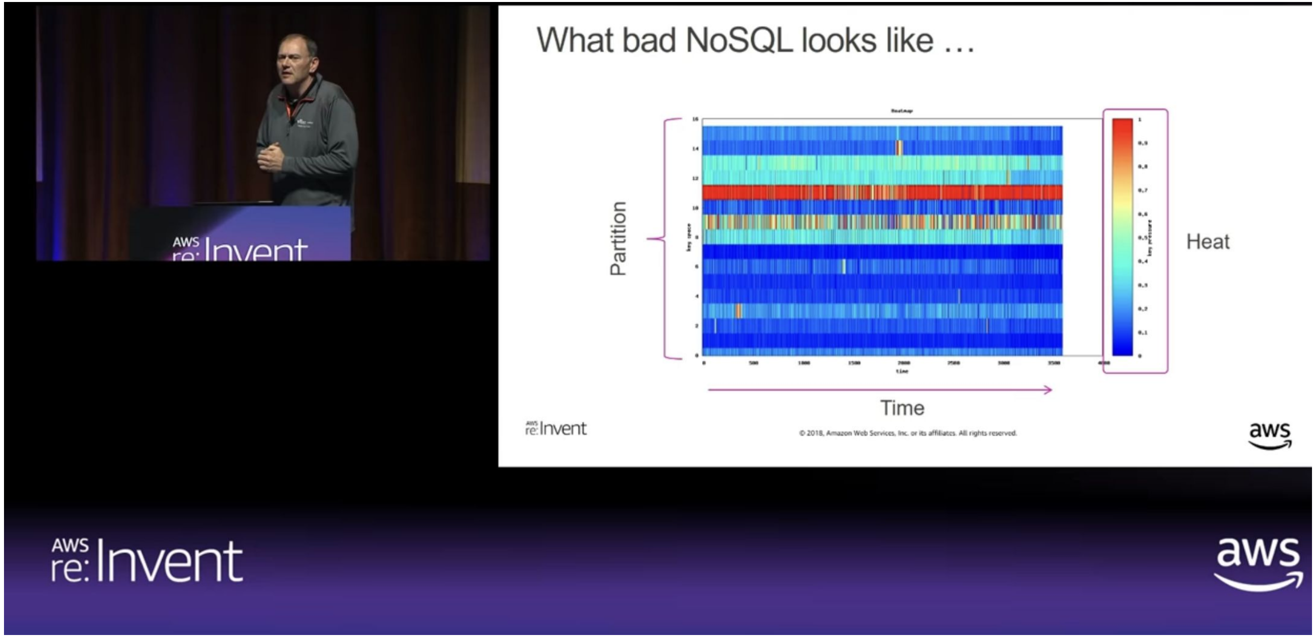
How It Works

- A Dynamo table is stored on a collection of nodes (partitions)
- Each table has a **partition key** and optional **sort key**, which uniquely identify all items in table
- **Partition key** decides which node owns the record
- **Sort key** determines how records are organized within a node
- Each item is collection of attributes, which can be scalar (string, number, binary, boolean) or complex types (list, map, set)

How It Works


Async replication
Partition 1
Partition 1 Secondary A
Partition 1 Secondary B
Hash partition key to find owning partition
Synchronous Replication
Client
Put Item
Request Router
Partition 2
Partition 2 Secondary A
Partition 2 Secondary B
Partition 3
Partition 3 Secondary A
Partition 3 Secondary B

Choosing Partition and Sort Keys



What bad NoSQL looks like …

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Choosing Partition and Sort Keys

- Key selection drives access patterns
- Partition key must be **high cardinality** to avoid hot partitions
  - Good: GUID, CustomerId
  - Bad: Status, Boolean
- Sort key can be used for **ordering** and modeling **1:n and n:n relationships**
  - Relationships modeled with composite keys
  - Order can be maintained with timestamps (updated_at) or sortable GUIDs, e.g. KSUID

Indexing

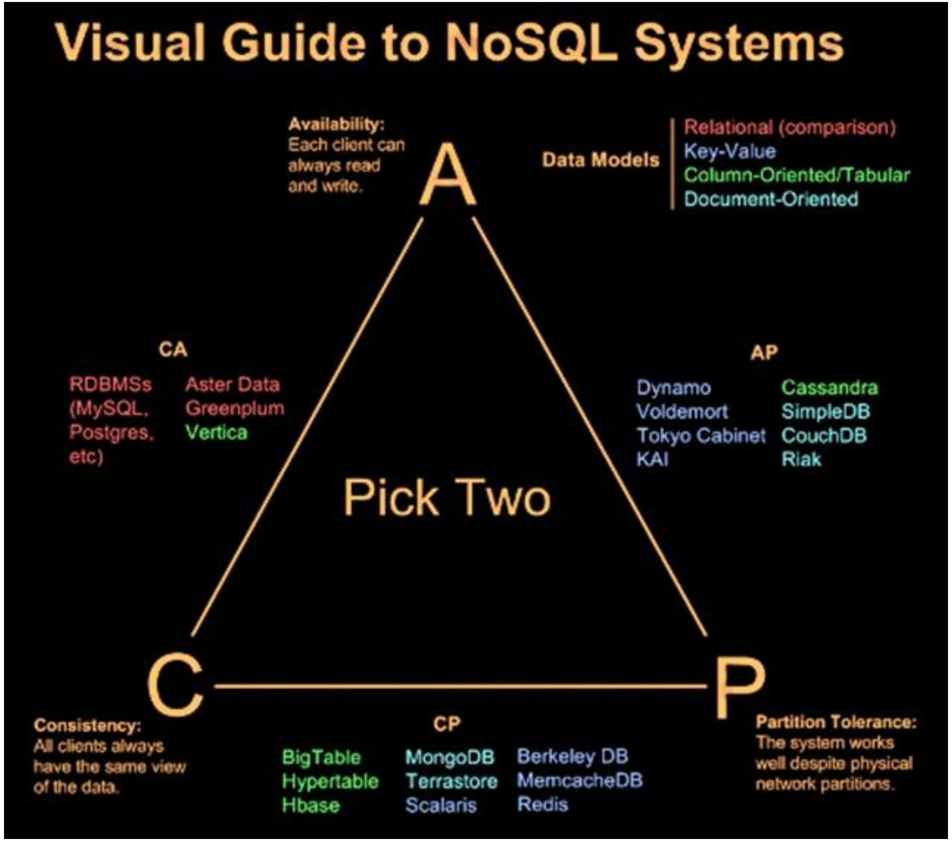
- Global secondary indexes (GSI) allow you to define a new partition key and sort key on the table
    - Enables new “views” on a table
    - RCU/WCU must be at least equal to table, or throttling may happen
    - GSI are only eventually consistent
- Local secondary indexes (LSI) allow you to define a new sort key on existing table partition key
    - Reorganizes data in a single partition. Imposes a 10 GB limit per hash key
    - Strongly consistent, as opposed to GSI

Consistency

- Writes always go to the owning node and are consistent
- Eventually consistent reads go to any partition
- Strongly consistent reads go to owning partition and cost 2x
- Prefer eventual consistency when possible
- **Only** table partition key / sort key and local secondary indexes can provide full read consistency
- TransactionWriteItems / TransactionGetItems can provide ACID compliance

Dynamo & CAP Theorem

- All distributed data stores can only provide 2 of 3
  - Consistency
  - Availability
  - Partition Tolerance
- Dynamo by default provides availability and partition tolerance
- Using strong consistency/transactions trades availability for consistency



Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

Consistency: All clients always have the same view of the data.

Partition Tolerance: The system works well despite physical network partitions.

Streams

- Provides a time-ordered sequence of item-level changes on a table
- Backed by Kinesis stream; out of the path of table requests
- Great for extending a Dynamo table with reactive functionality

Note on Capacity

- Billed by read capacity units (RCU) and write capacity units (WCU)
- Either on demand or provisioned modes
- Tip: Until your Dynamo workload is known, use on demand
- Cost of over-provisioning will likely exceed on demand costs

Limits

- Up to 40,000 RCU/WCU per table
- Max partition key size 2048 bytes
- Max sort key size is 1024 bytes
- Max 400KB item size
- Only 1MB of table data scanned per query before filters applied
- Single partition can only have 3000 RCU / 1000 WCU
    - In other words, a key item cannot be written to > 1000 WCU
- Local secondary index can only contain 10GB of data per partition key
- 20 GSIs per table
- 5 LSIs per table

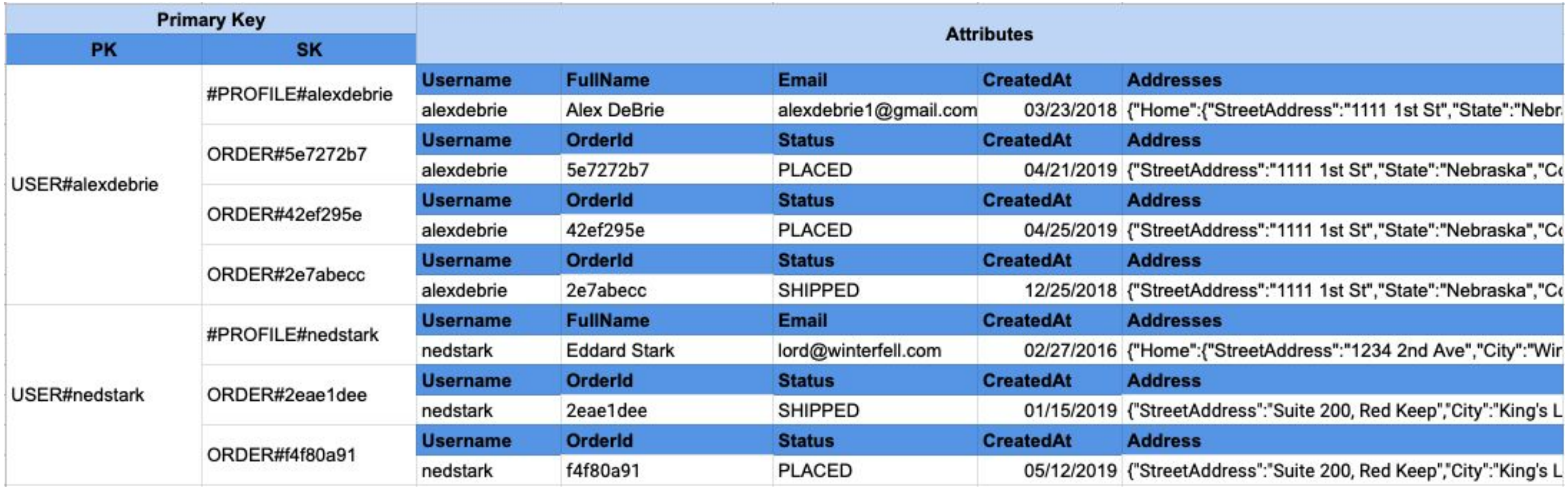
Single Table Design

- Different entities can & should live in the same Dynamo table
  - 1 table per entity (e.g. Users table, Roles table, Org table) is often an antipattern
  - Attempting to join across Dynamo tables can kill performance
- Solution: “pre-join” records of different types into a single table
- Partition keys allow us to specify “item collections”
- Sort keys allow us to define relationships between items
- GSIs / LSIs give us additional querying flexibility

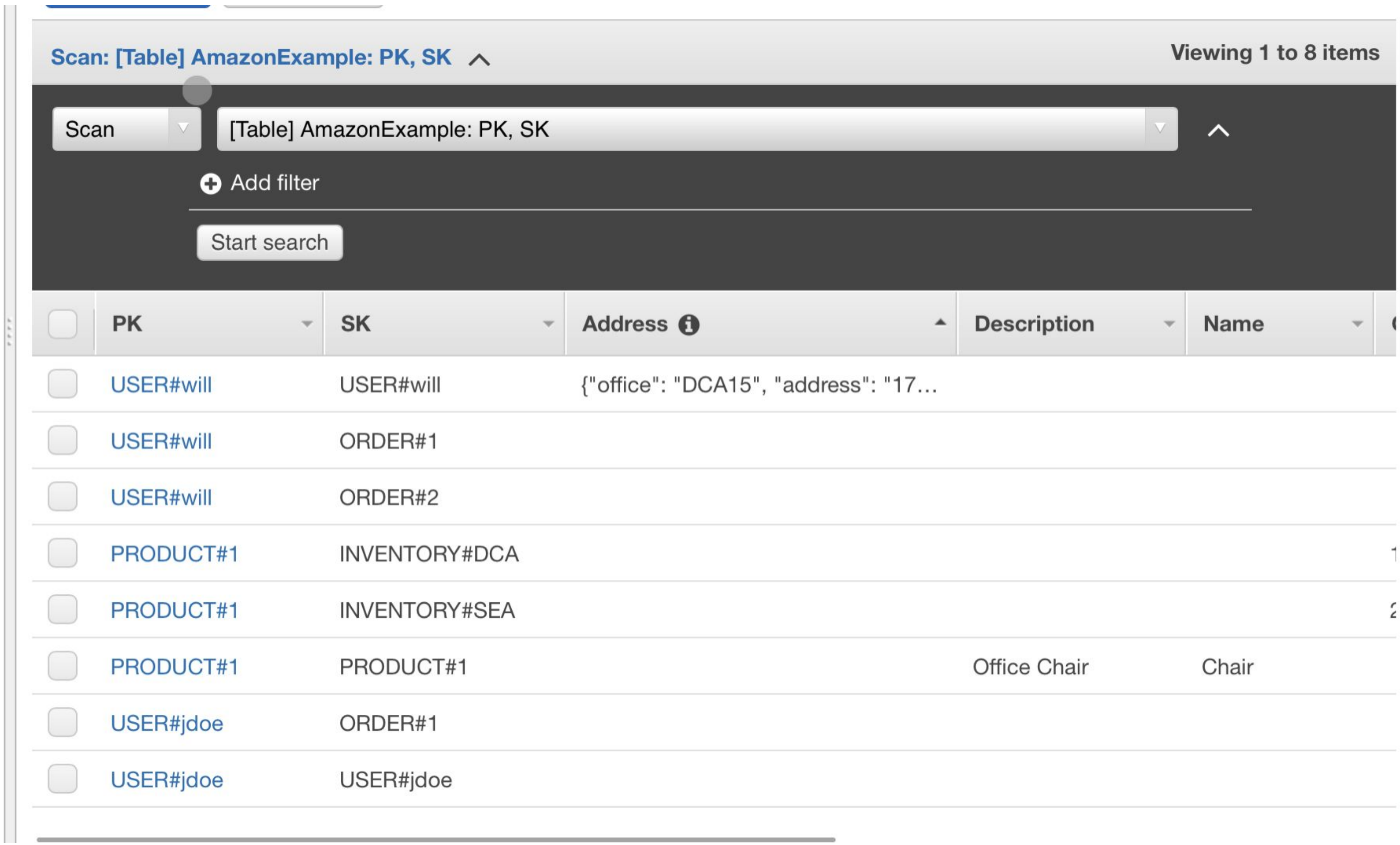
Dynamo Patterns

Single Table Design Example

- Usecase: modeling simple e-commerce site
- Entities: User, Order, Product, Inventory



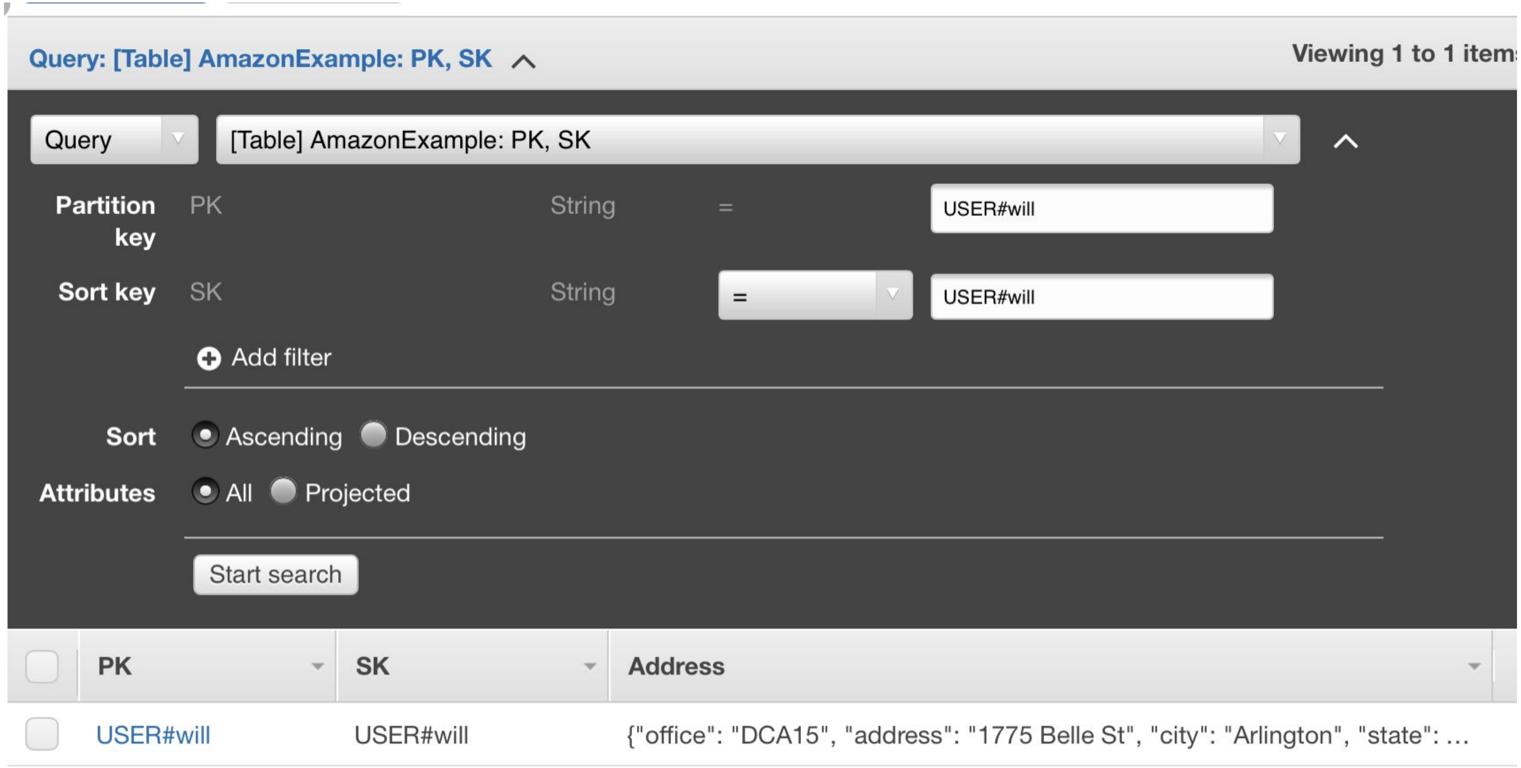
Primary Key		Attributes				
PK	SK					
USER#alexdebrie	#PROFILE#alexdebrie	Username	FullName	Email	CreatedAt	Addresses
		alexdebrie	Alex DeBrie	alexdebrie1@gmail.com	03/23/2018	{"Home":{"StreetAddress":"1111 1st St","State":"Nebr
	ORDER#5e7272b7	Username	OrderId	Status	CreatedAt	Address
		alexdebrie	5e7272b7	PLACED	04/21/2019	{"StreetAddress":"1111 1st St","State":"Nebraska","C
	ORDER#42ef295e	Username	OrderId	Status	CreatedAt	Address
		alexdebrie	42ef295e	PLACED	04/25/2019	{"StreetAddress":"1111 1st St","State":"Nebraska","C
	ORDER#2e7abecc	Username	OrderId	Status	CreatedAt	Address
		alexdebrie	2e7abecc	SHIPPED	12/25/2018	{"StreetAddress":"1111 1st St","State":"Nebraska","C
USER#nedstark	#PROFILE#nedstark	Username	FullName	Email	CreatedAt	Addresses
		nedstark	Eddard Stark	lord@winterfell.com	02/27/2016	{"Home":{"StreetAddress":"1234 2nd Ave","City":"Wi
	ORDER#2eae1dee	Username	OrderId	Status	CreatedAt	Address
		nedstark	2eae1dee	SHIPPED	01/15/2019	{"StreetAddress":"Suite 200, Red Keep","City":"King's L
	ORDER#f4f80a91	Username	OrderId	Status	CreatedAt	Address
		nedstark	f4f80a91	PLACED	05/12/2019	{"StreetAddress":"Suite 200, Red Keep","City":"King's L



Scan: [Table] AmazonExample: PK, SK

Viewing 1 to 8 items

□	PK	SK	Address	Description	Name	
□	USER#will	USER#will	{"office": "DCA15", "address": "17…			
□	USER#will	ORDER#1				
□	USER#will	ORDER#2				
□	PRODUCT#1	INVENTORY#DCA				1
□	PRODUCT#1	INVENTORY#SEA				2
□	PRODUCT#1	PRODUCT#1		Office Chair	Chair	
□	USER#jdoe	ORDER#1				
□	USER#jdoe	USER#jdoe				



Query: [Table] AmazonExample: PK, SK

Viewing 1 to 1 item

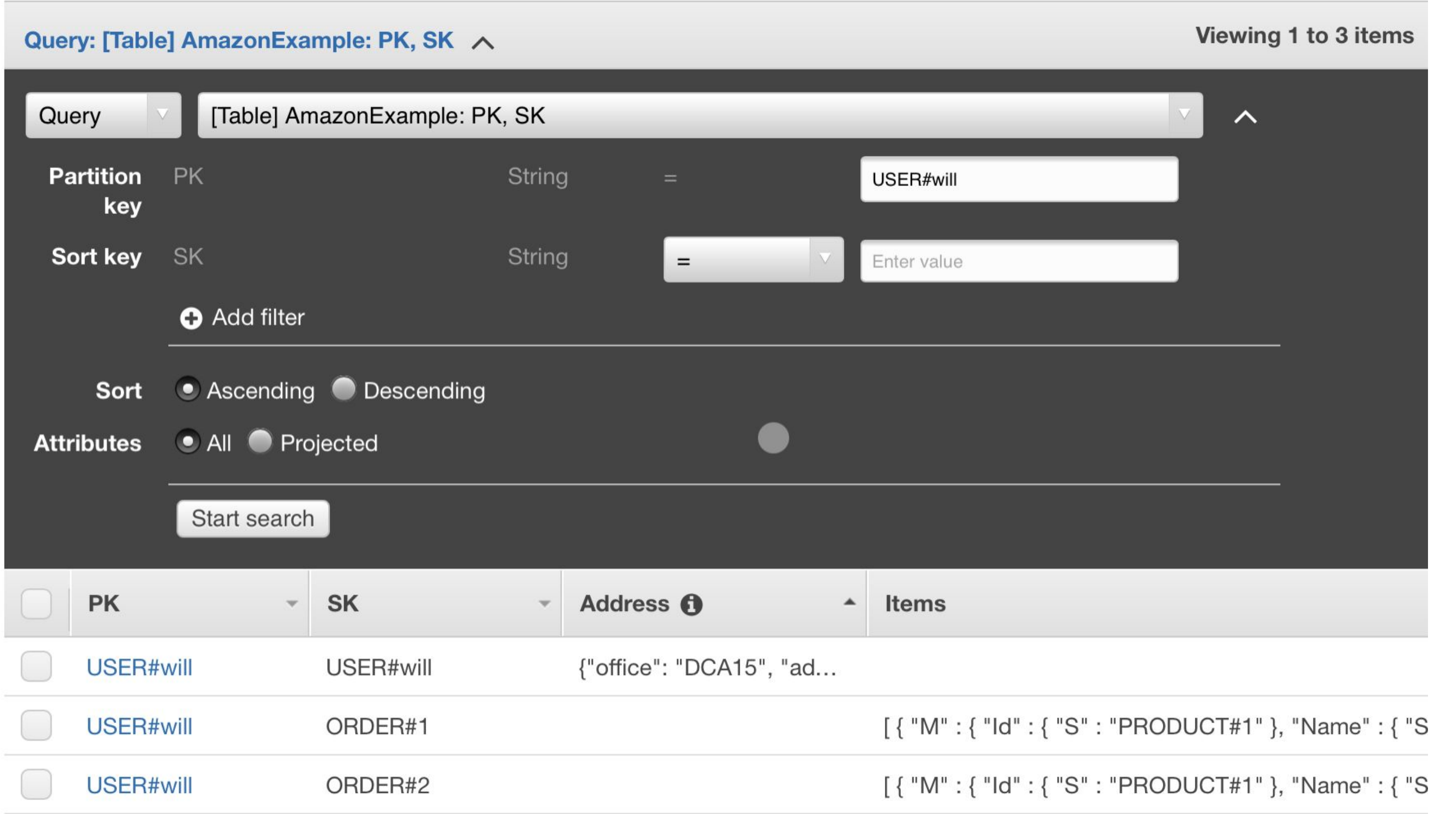
Partition key PK String = USER#will

Sort key SK String = USER#will

Sort ● Ascending ○ Descending

Attributes ● All ○ Projected

□	PK	SK	Address
□	USER#will	USER#will	{"office": "DCA15", "address": "1775 Belle St", "city": "Arlington", "state": …



Query: [Table] AmazonExample: PK, SK

Viewing 1 to 3 items

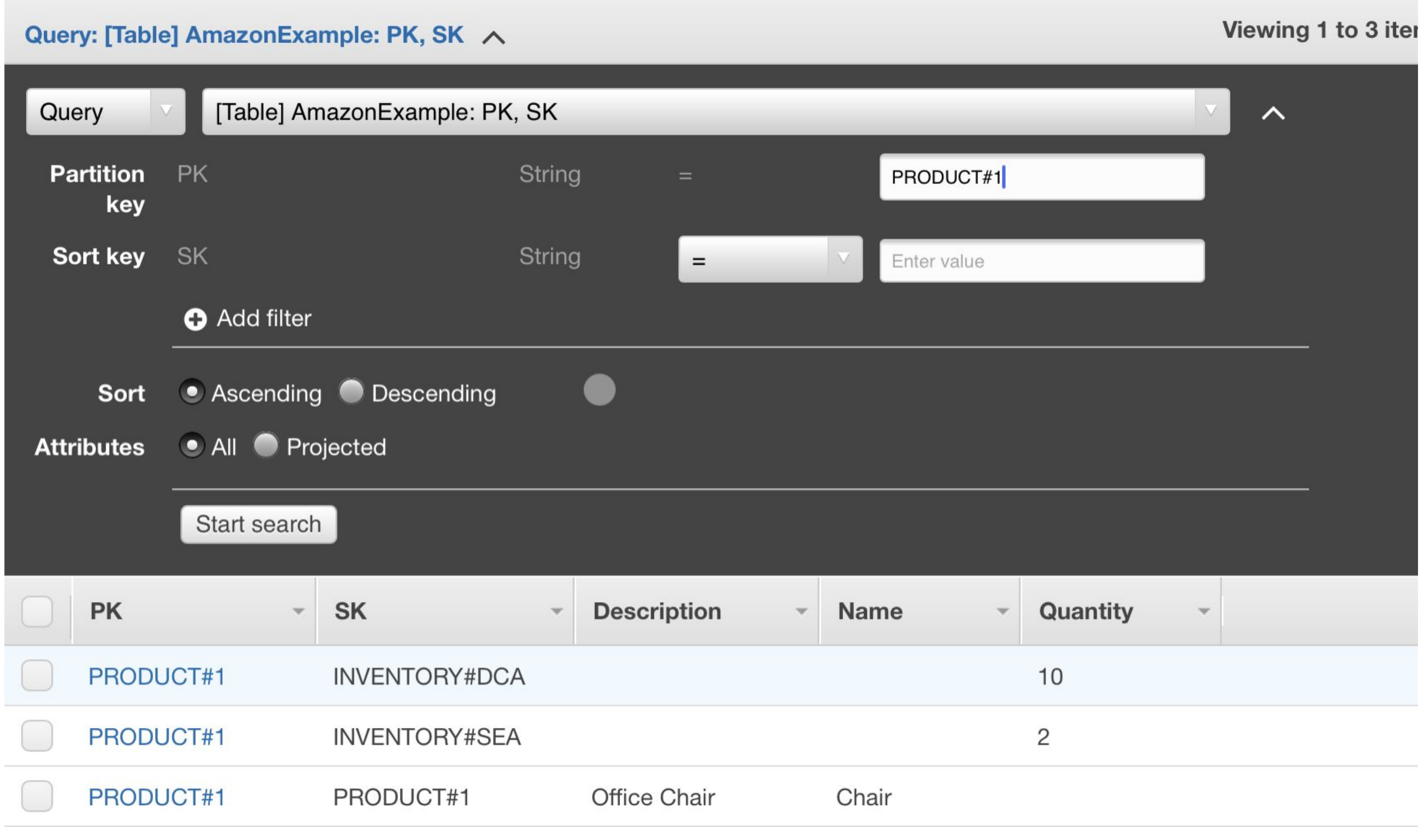
Partition key PK String = USER#will

Sort key SK String = Enter value

Sort ● Ascending ○ Descending

Attributes ● All ○ Projected

□	PK	SK	Address	Items
□	USER#will	USER#will	{"office": "DCA15", "ad…	
□	USER#will	ORDER#1		[{ "M" : { "Id" : { "S" : "PRODUCT#1" }, "Name" : { "S
□	USER#will	ORDER#2		[{ "M" : { "Id" : { "S" : "PRODUCT#1" }, "Name" : { "S



Query | [Table] AmazonExample: PK, SK

Partition key PK String = PRODUCT#1

Sort key SK String = Enter value

Sort ● Ascending ○ Descending

Attributes ● All ○ Projected

□	PK	SK	Description	Name	Quantity
□	PRODUCT#1	INVENTORY#DCA			10
□	PRODUCT#1	INVENTORY#SEA			2
□	PRODUCT#1	PRODUCT#1	Office Chair	Chair	

Single Table Design Tips

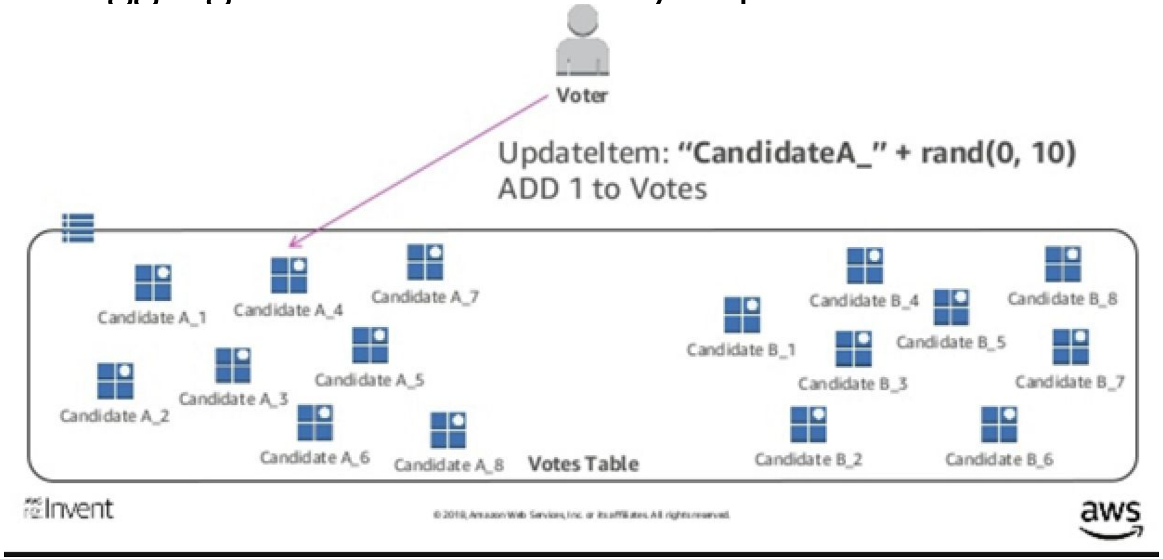
- Define access patterns up-front
- Don’t lock partition key and sort key into one usecase
- Attempt to de-normalize data where possible to reduce number of queries
- Leverage sortable IDs to maintain order
- Try to keep partition and sort key identifiers short to prevent hitting size limits

Uniqueness Constraints on Multiple Attributes

- Uniqueness constraints can be added with new entity types
- Example: enforcing uniqueness on both email and user ID attribute
- Create write transaction
    - Item 1 {PK: “USER#3921”, SK: “USER#3921”}
    - Item 2 {PK: “EMAIL#will@gmail.com”, SK: “EMAIL#will@gmail.com”}
    - Conditional check on neither existing
- If either exists, the transaction fails

Avoiding Hot Keys

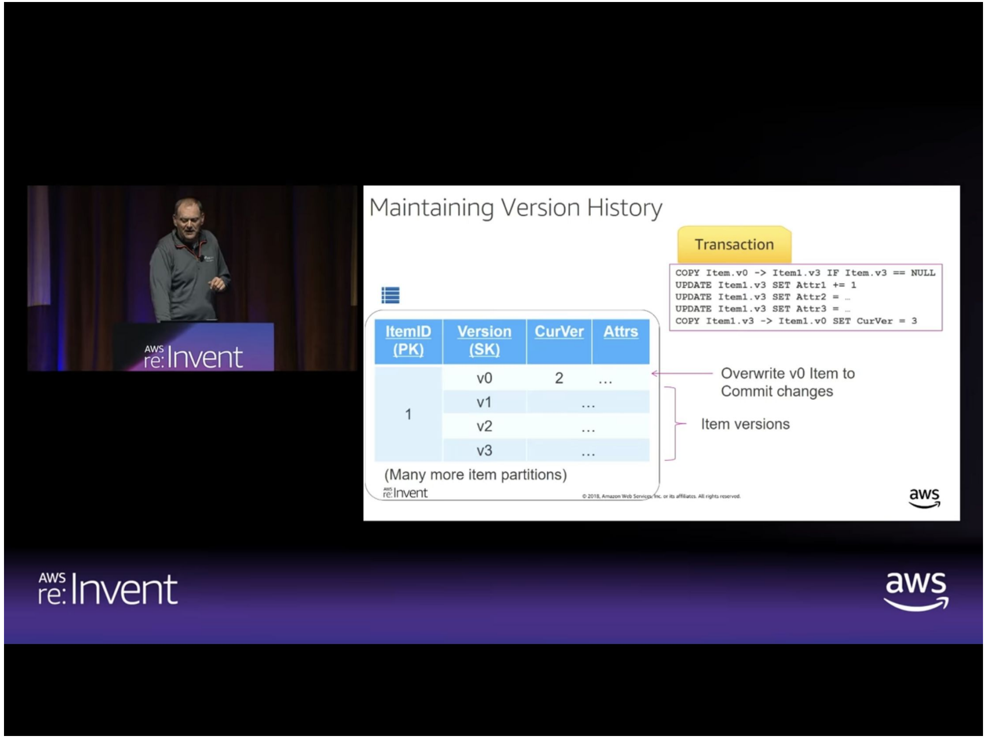
- Example: writing many votes to a candidate record
- To avoid high WCU to one item, shard the item among many records, and compute an aggregate count via an async process



UpdateItem: **"CandidateA_" + rand(0, 10)**
ADD 1 to Votes

Maintaining Item History

- Using version tag as sort key allows maintaining of write history
- Use the following write pattern:
  - Fetch item ID = X, version = v0
  - In write transaction,
  - Set previous v0 as new vX item
  - Update attributes on v0
- v0 always contains latest record



Maintaining Version History

Transaction

```
COPY Item.v0 -> Item1.v3 IF Item.v3 == NULL
UPDATE Item1.v3 SET Attr1 += 1
UPDATE Item1.v3 SET Attr2 = …
UPDATE Item1.v3 SET Attr3 = …
COPY Item1.v3 -> Item1.v0 SET CurVer = 3
```

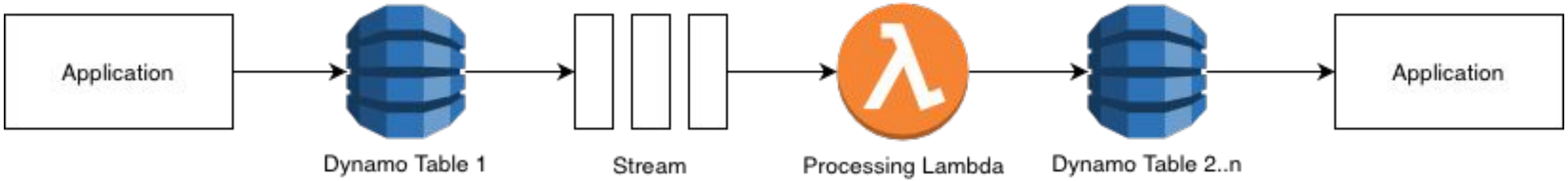
ItemID (PK)	Version (SK)	CurVer	Attrs
1	v0	2	…
	v1		…
	v2		…
	v3		…

(Many more item partitions)

Overwrite v0 Item to Commit changes

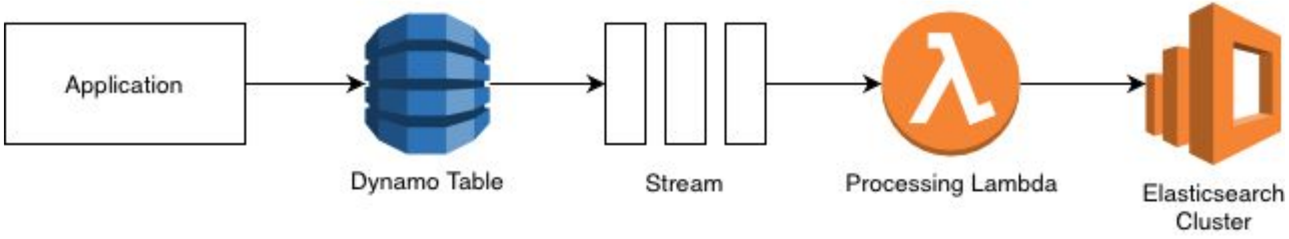
Item versions

Aggregation with Streams



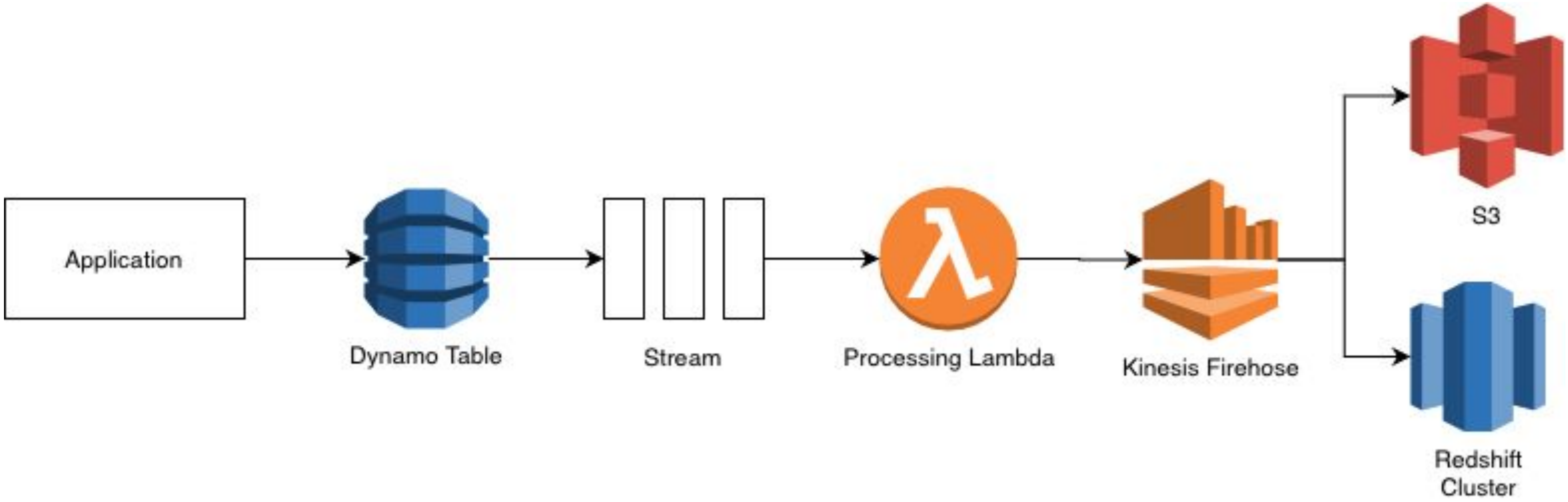
- Read updates from stream, and push metadata / aggregations back to Dynamo
- Example: on new Order item, update User record openOrders += 1

Full Text Search with Streams



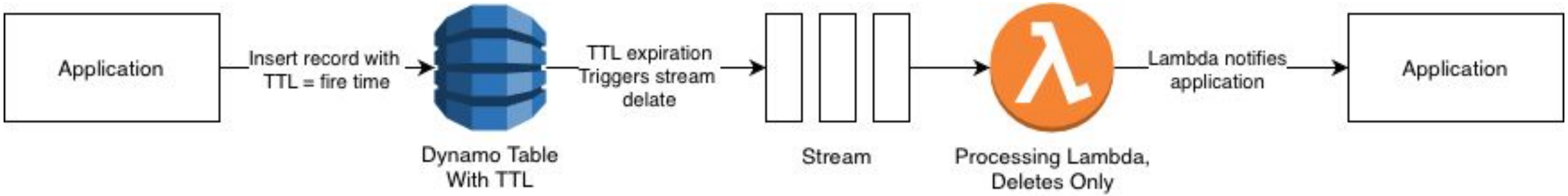
- Transform and push items into ElasticSearch to enable full-text search
- Maintains Dynamo as source of truth, but enables more powerful querying options

Load to Warehouse with Streams



- Push items into a data warehouse (e.g. S3, Redshift) to enable flexible BI querying

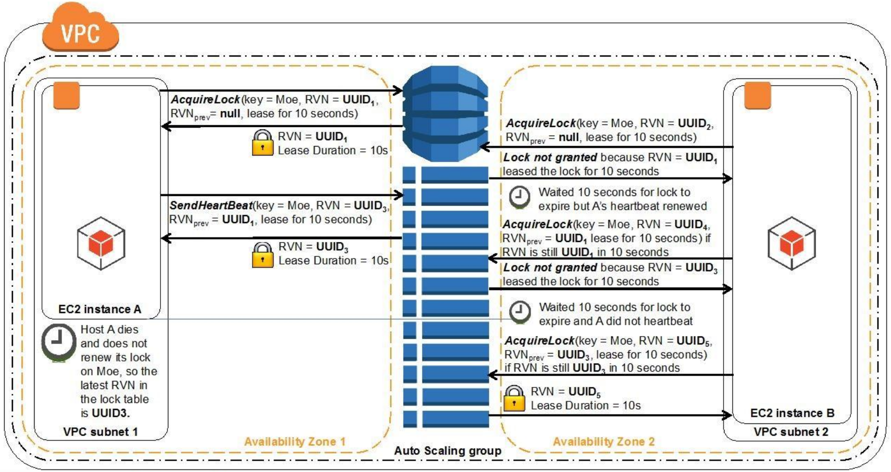
Scheduling with Item TTL


Application
Insert record with TTL = fire time
Dynamo Table With TTL
TTL expiration Triggers stream delate
Stream
Processing Lambda, Deletes Only
Lambda notifies application
Application

- Documents in Dynamo can define a time-to-live attribute. This is helpful for caching, leases, and scheduling
- Can schedule events by pushing an item with a TTL at desired fire date
- Listen for delete events, and notify application
- TTL is implemented as a background scanning process on each partition. Depending on table usage, delete could take as long as 48h to process. Typically is much shorter

Distributed Locking

- Quickly build distributed lock with open-source Amazon DynamoDB Lock Client
- Use cases: making sure two workers don’t operate on same entity, leader election
- Only requires a Dynamo DB table with partition key “key”
- Supports heartbeats, lease duration, blocking / non-blocking lock acquisitions



AcquireLock(key = Moe, RVN = **UUID$_1$**, RVN_{prev}= **null**, lease for 10 seconds)

RVN = **UUID$_1$**
Lease Duration = 10s

SendHeartBeat(key = Moe, RVN = **UUID$_3$**, RVN_{prev} = **UUID$_1$**, lease for 10 seconds)

RVN = **UUID$_3$**
Lease Duration = 10s

Host A dies and does not renew its lock on Moe, so the latest RVN in the lock table is **UUID3.**

AcquireLock(key = Moe, RVN = **UUID$_2$**, RVN_{prev} = **null**, lease for 10 seconds)

Lock not granted because RVN = **UUID$_1$** leased the lock for 10 seconds

Waited 10 seconds for lock to expire but A’s heartbeat renewed

AcquireLock(key = Moe, RVN = **UUID$_4$**, RVN_{prev} = **UUID$_1$** lease for 10 seconds) if RVN is still **UUID$_1$** in 10 seconds

Lock not granted because RVN = **UUID$_3$** leased the lock for 10 seconds

Waited 10 seconds for lock to expire and A did not heartbeat

AcquireLock(key = Moe, RVN = **UUID$_5$**, RVN_{prev} = **UUID$_3$**, lease for 10 seconds) if RVN is still **UUID$_3$** in 10 seconds

RVN = **UUID$_5$**
Lease Duration = 10s

Optimistic Locking

- Java library has @DynamoDBVersionAttribute annotation, which sets up optimistic item locking
- Each item put is given a conditional check, where current version = expected version
- Write will reject if versions do not match
- Make sure to handle ConditionalCheckFailed runtime exceptions

Serverless Dynamo Frontends

- AWS API Gateway can add REST endpoints on top of Dynamo tables using service proxies
    - No-code solution
    - APIs can be versioned
    - SIGv4 Authentication
    - E.g. map route GET /companies/Amazon/employees/1 to query PK: Amazon, SK: 1; transform and return result
- AWS AppSync can add GraphQL operations on top of Dynamo tables

From Millisecond to Microsecond

- Dynamo Accelerator (DAX) is fully managed caching solution which brings latencies down to microseconds
  - In-memory cache of items and queries
  - Only supports eventually consistent reads
- Global tables can replicate Dynamo tables cross-region
  - Multi-master replicas
  - Writes propagated cross-region within a second
  - Last-writer-wins for cross-region write conflicts

Handling Migrations

- If possible, handle new attribute defaults in business logic
- For small backfills / migrations, scripts are preferable
  - Parallel scans possible with TotalSegments / SegmentNumber arguments
- For large backfills / migrations, use EMR
  - AWS EMR has built in Dynamo adapters
  - Load a Dynamo table into Hive, make change, then load back to Dynamo
  - Make sure to set a % of capacity to use during job

Resources

- ReInvent 2018 DAT401 - Advanced Design Patterns for DynamoDB https://youtu.be/HaEPXoXVf2k
- The DynamoDB Book by Alex Debrie https://www.dynamodbbook.com
- Advanced Design Patterns for Amazon DynamoDB by National Australia Bank https://link.medium.com/ypcCdKt6Kbb
- Dynamo Docs https://docs.aws.amazon.com/dynamodb/index.html