Algorithmic Methods of Data Mining

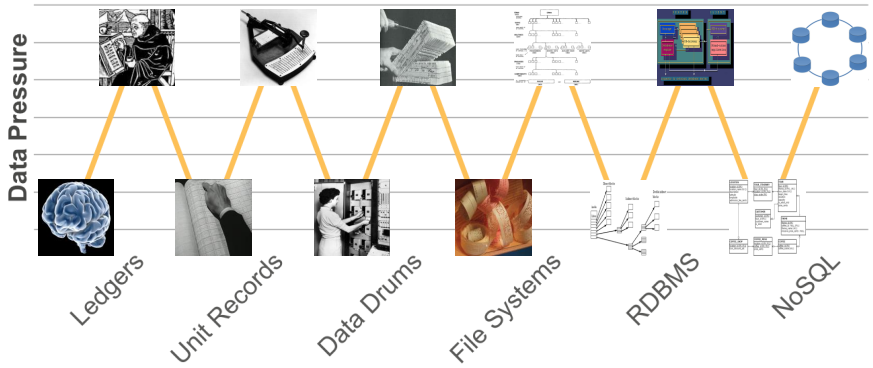
DynamoDB

Ioannis Chatzigiannakis

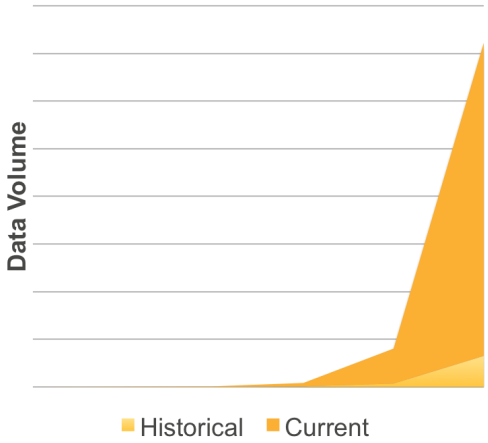
Sapienza University of Rome

Laboratory 5

Timeline of Database Technology

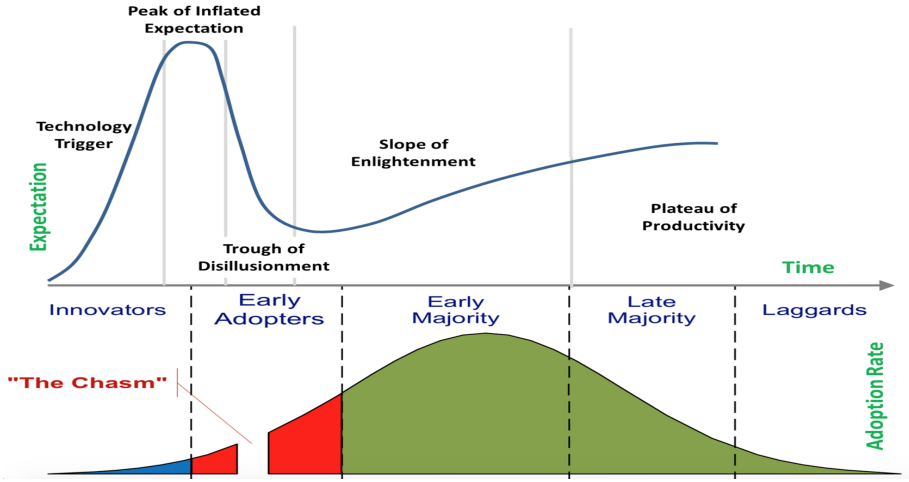


Data Volume Since 2010



- 90% of stored data generated in last 2 years
- 1 Terabyte of data in 2010 equals 6.5 Petabytes today
- Linear correlation between data pressure and technical innovation
- No reason these trends will not continue over time

Technology Adoption and the Hype Curve



AWS: DynamoDB

- A NoSQL key-value and document database.
  - Key-Value Database:
    - Pairs of Key-Values (records) are stored into the same namespace.
    - Values can contain any type of records.
    - The simples possible data model.
  - Document Database:
    - Documents (records) are organized into groups called collections.
    - Collections $\sim$ Tables of RDBMS.
    - Can be viewed as an extension of the Key-Value database.
- Fully Managed Service.
  - Support databases of virtually any size.
  - Scale to more than 10 trillion requests per day.
  - Service request peaks greater than 20 million requests per second.

Some Use Cases

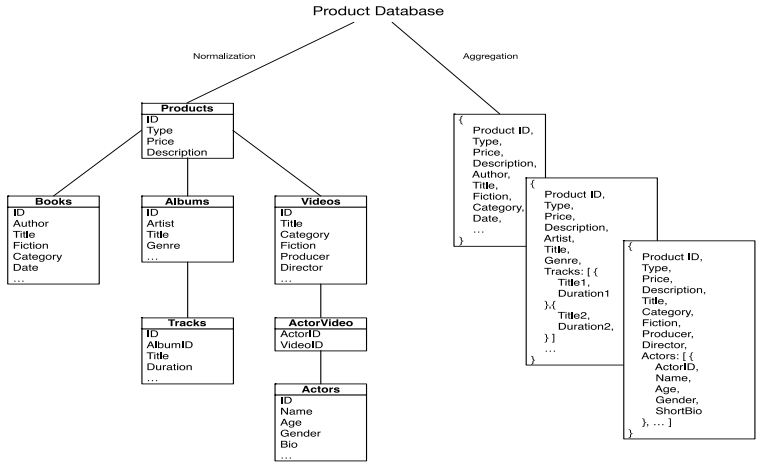
- Netflix – uses DynamoDB to run A/B testing that builds personalized streaming experiences for their 125+ million customers.
- United States Census 2020 – uses DynamoDB to scale response collection on mobile or desktop, allowing people to participate in its decennial count online for the first time.
- Samsung Electronics – uses DynamoDB for their petabyte-sized mobile app backups, resulting in consistent high performance and cost savings.
- ...

Why NoSQL?

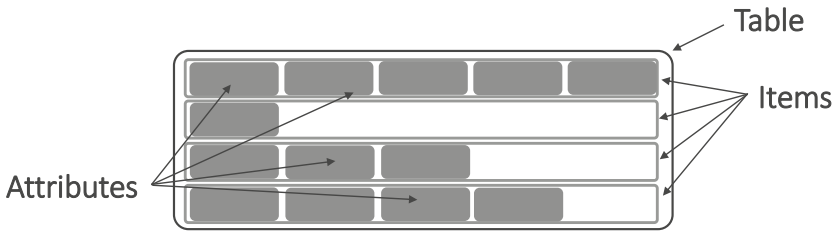
SQL NoSQL

Optimized for storage	Optimized for compute
Normalized/relational	Denormalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLAP	Built for OLTP at scale

SQL vs. NoSQL Access Pattern



Table



Partition Key

Sort Key

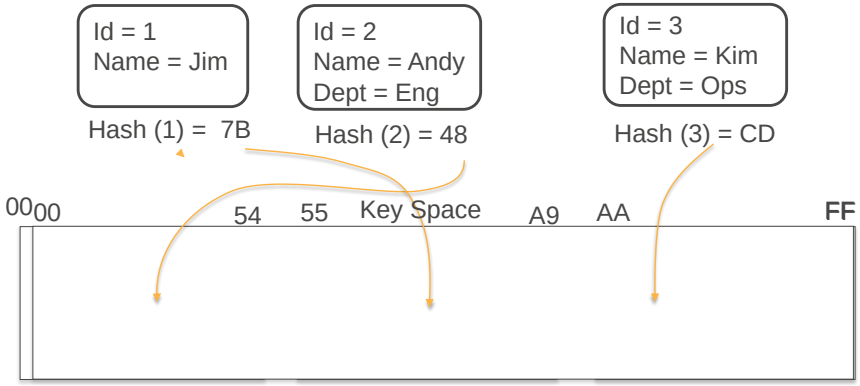
Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities

All items for key
==, <, >, >=, <=
"begins with"
"between"
"contains"
"in"
sorted results
counts
top/bottom N values

Partition Keys

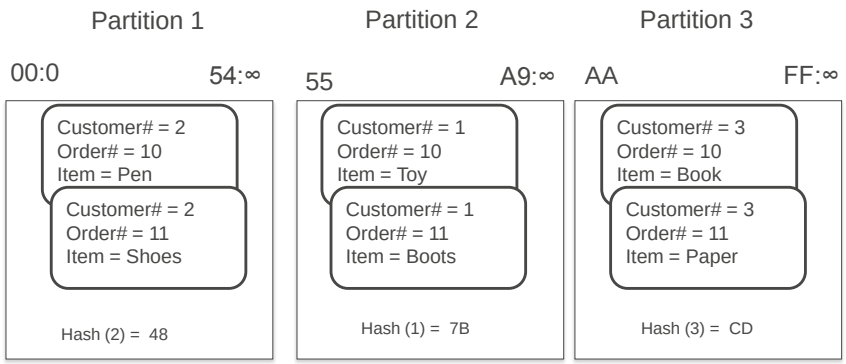
Partition Key uniquely identifies an item
Partition Key is used for building an unordered hash index
Allows table to be partitioned for scale



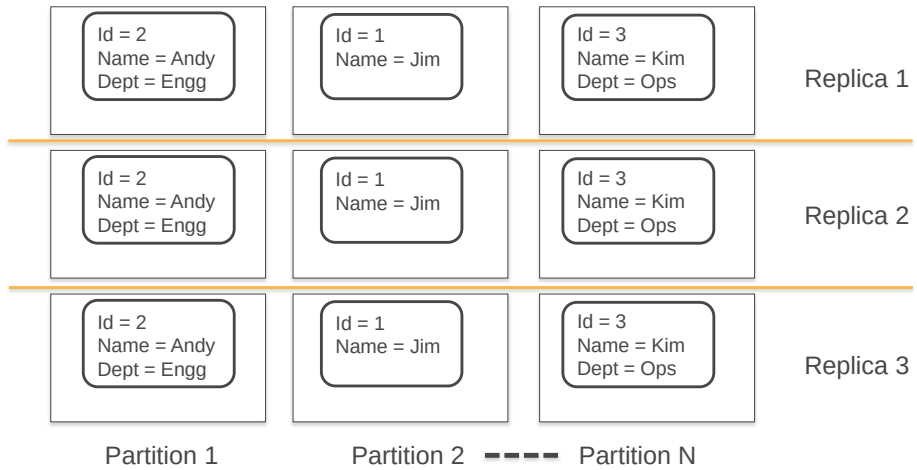
Partition:Sort Key

Partition:Sort Key uses two attributes together to uniquely identify an Item
Within unordered hash index, data is arranged by the sort key
No limit on the number of items (∞) per partition key

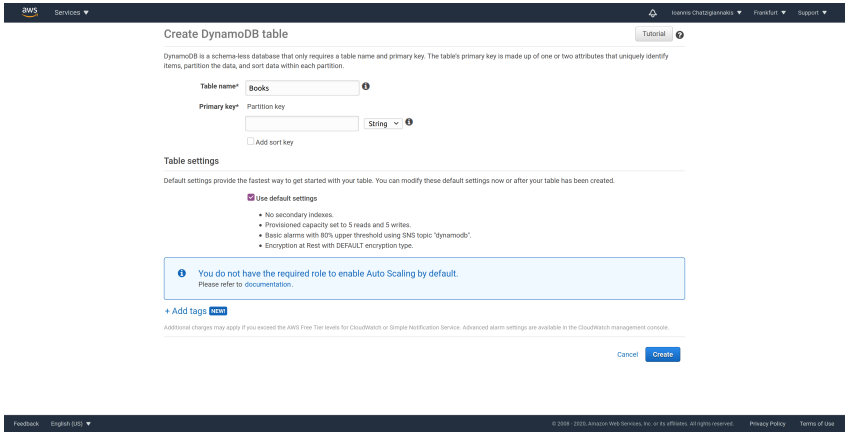
- Except if you have local secondary indexes



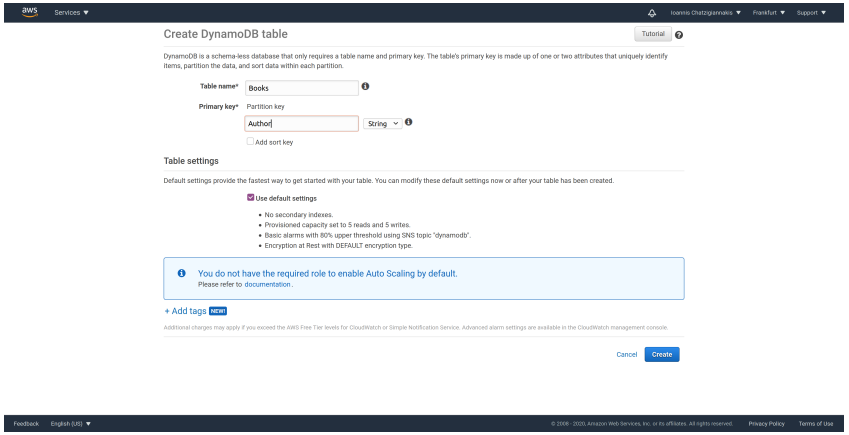
Partitions are three-way replicated



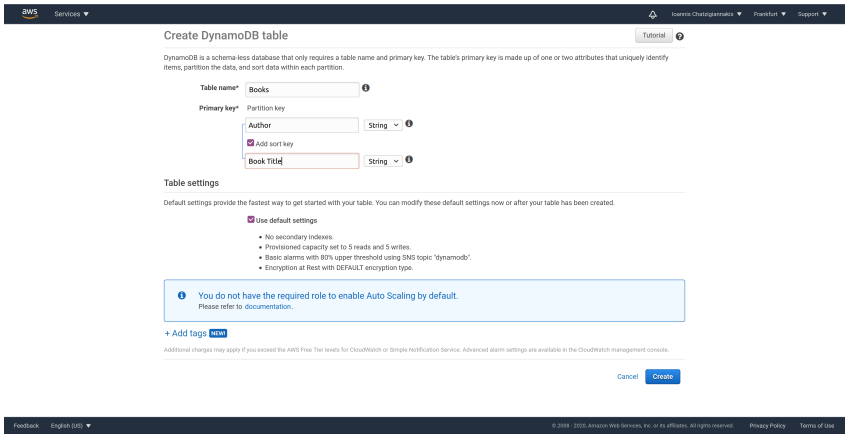
Choose Table Name



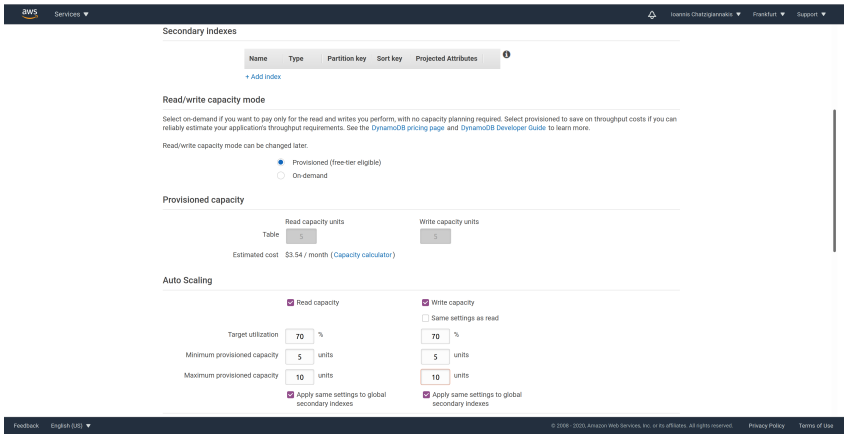
Define Primary Key



Add Sort Key



Select Read/Write Capacity Mode



Roles and Encryption

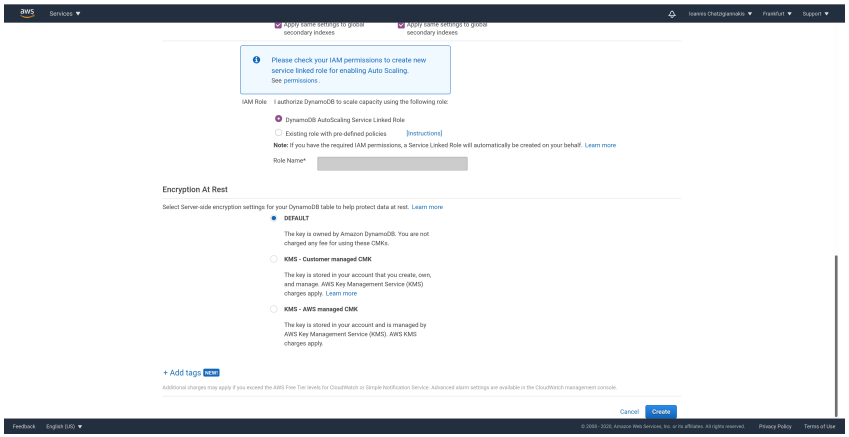
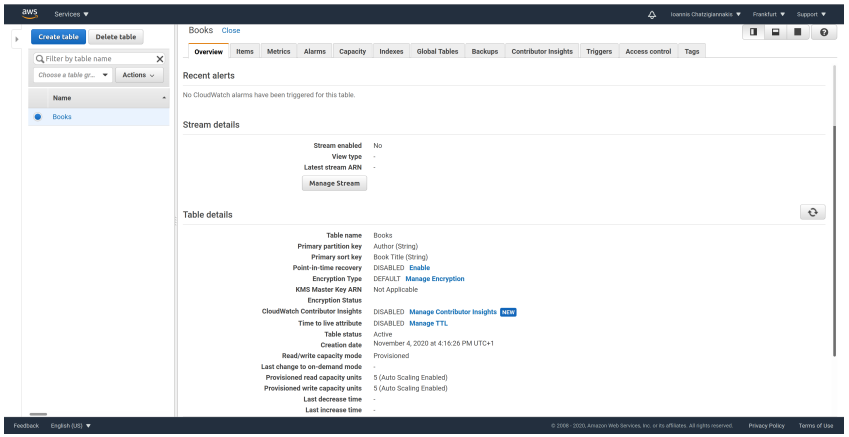
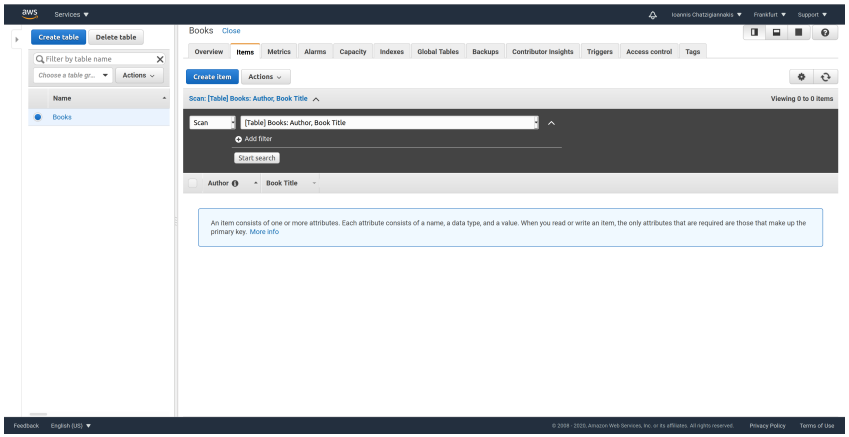


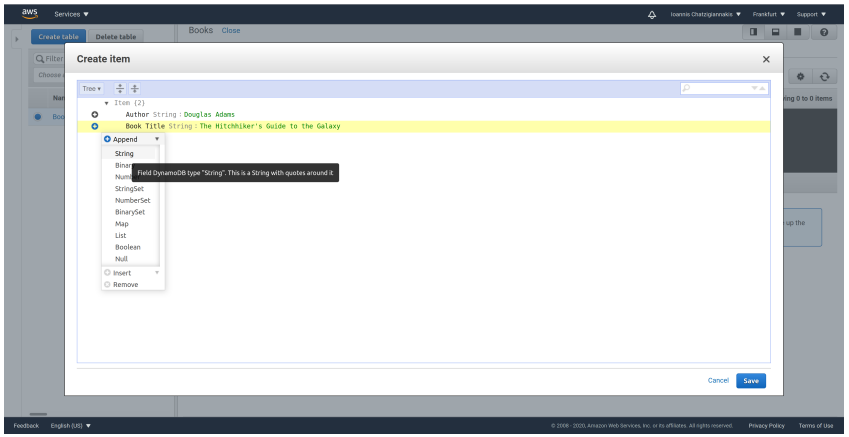
Table Overview



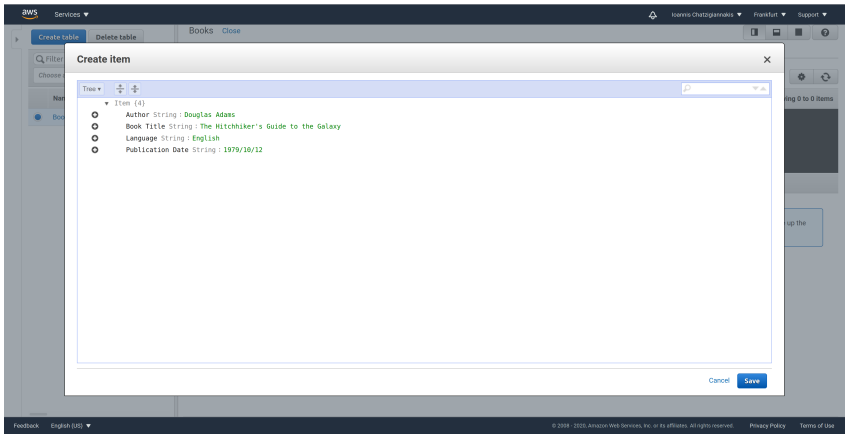
Create Item



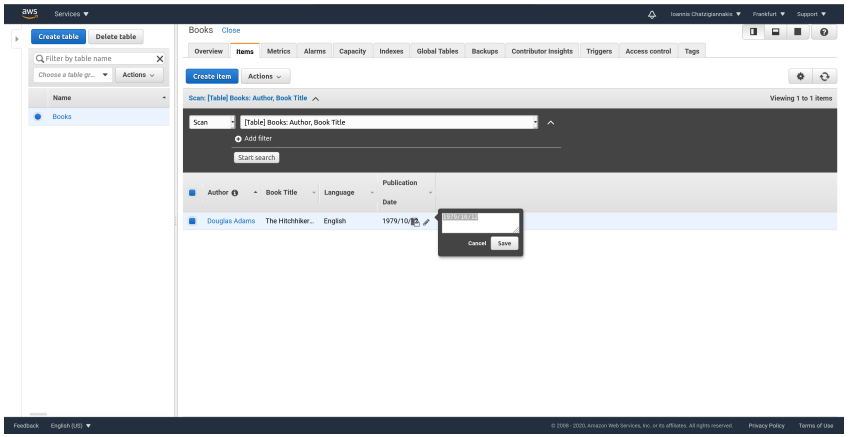
Add New Values



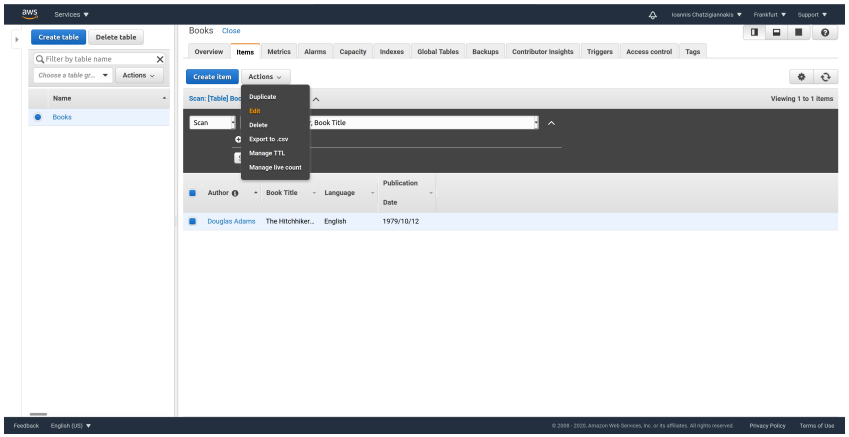
Save Record



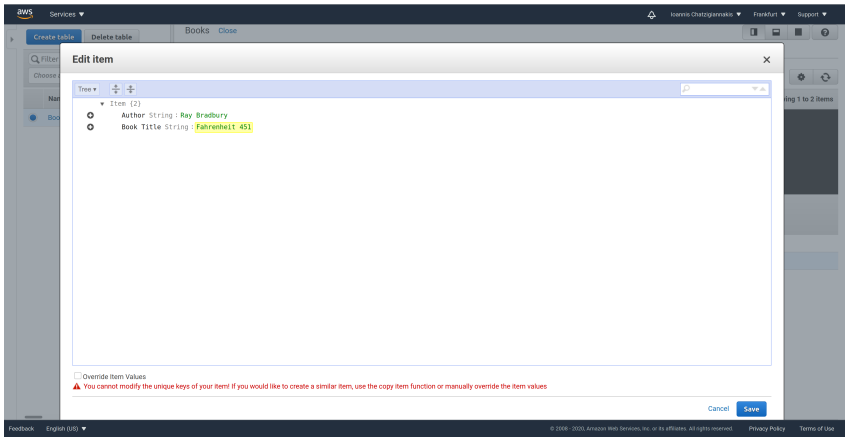
Edit Values



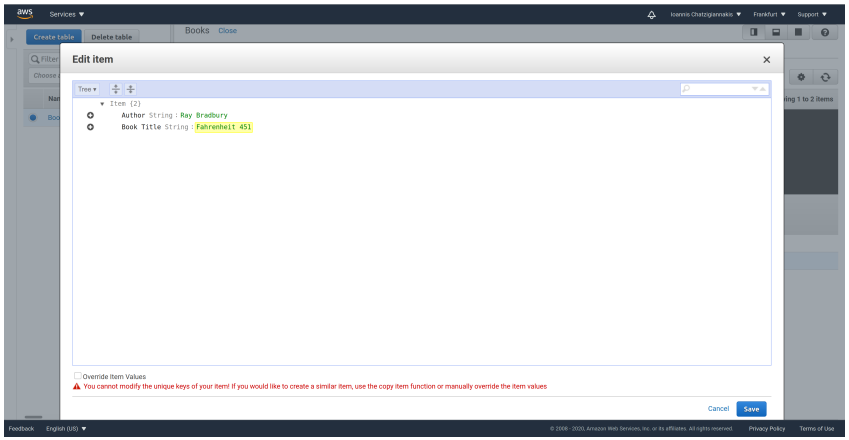
Edit Record



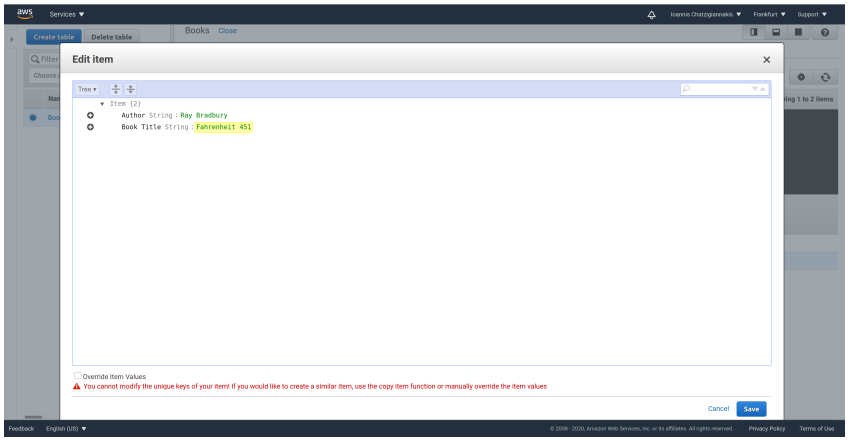
Modify Keys



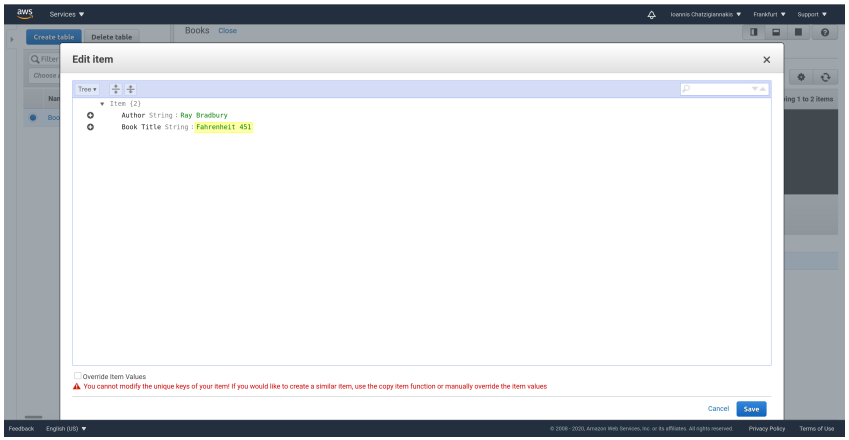
Instance Ready to Launch



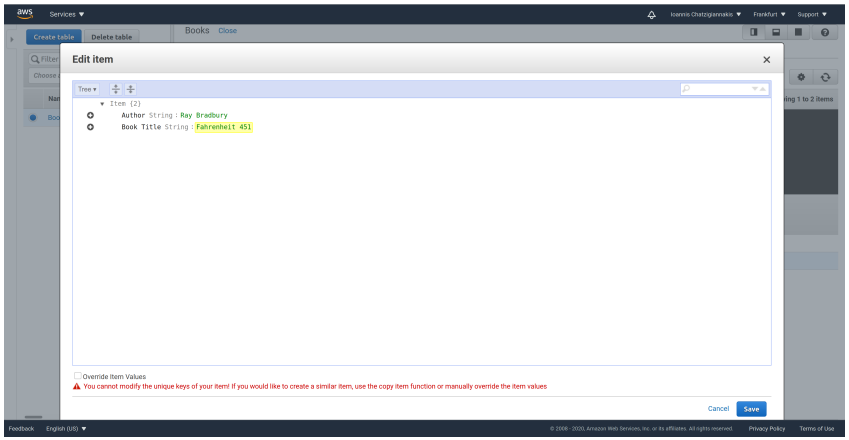
List of Instances



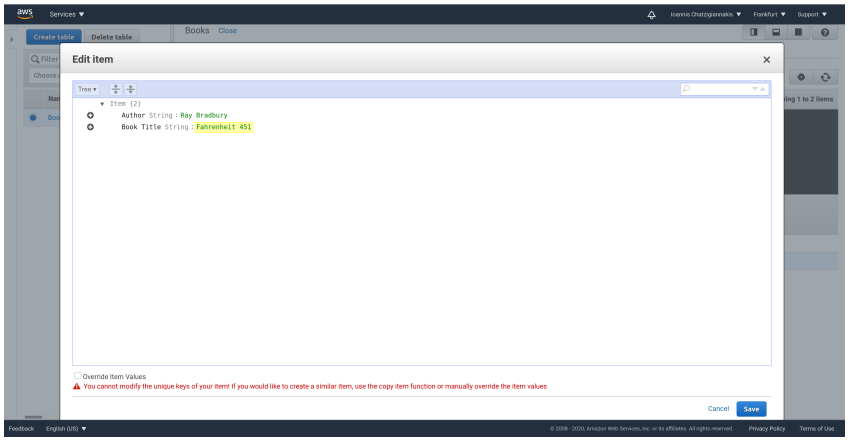
Overview of Instance



Connect to Instance



Command Line Console



Scaling

- Throughput
  - Provision any amount of throughput to a table.
  - Write capacity units (WCUs) – measured in 1 KB per second.
  - Read capacity units (WCUs) – measured in 4 KB per second.
    - RCUs measure strictly consistent reads.
    - Eventually consistent reads cost $\frac{1}{2}$ of consistent reads.
  - Read and write throughput limits are independent.
- Size
  - Add any number of items to a table.
  - Maximum item size is 400 KB.
- Scaling is achieved through partitioning.
  - $Size = \frac{Total\ Size}{10GB}$
  - $Capacity = \frac{Total\ RCU}{3000} + \frac{Total\ WCU}{1000}$
  - $TotalPartitions = \lceil max(Size, Capacity) \rceil$

Scaling Example

You are storing book data – on average a book record requires 80KB of data. Additionally, a lot of people would like to retrieve this game data and you expect about 1800 eventually consistent reads per second.

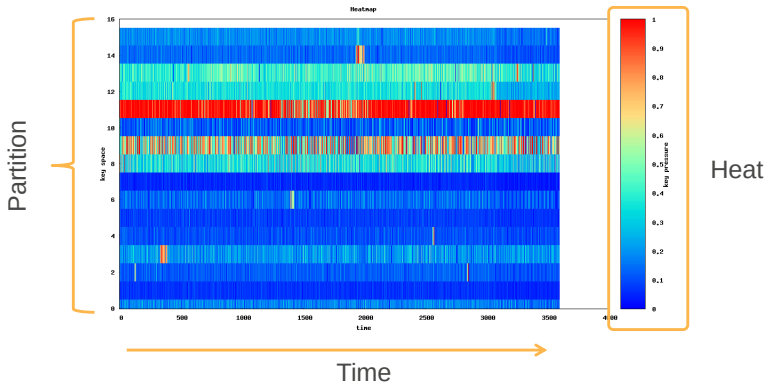
- How many WCUs are needed to write 400 books per second ?
- How many RCUs are needed for 1800 strictly consistent reads per second?
- How many RCUs are needed for 2400 eventually consistent reads per second?

Partitions Example

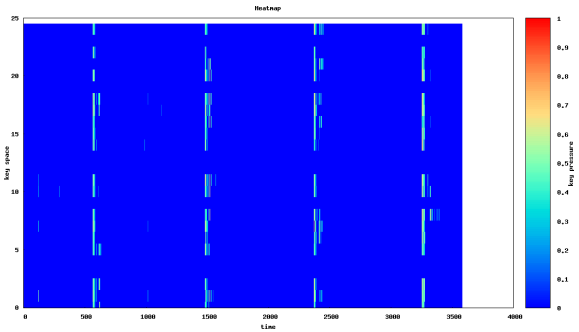
You are storing 180GB of book data – with 36000 RCUs and 32000 WCUs.

- $Size = \frac{180}{10=18}$
- $Capacity = \frac{36000}{3000} + \frac{32000}{1000} = 12 + 32$
- $TotalPartitions = \lceil max(18, 44) \rceil = 44$
- RCUs and WCUs are uniformly spread across partitions:
  - $4.09GB$ per partition.
  - 818.18 RCUs per partition.
  - 727.27 WCUs per partition.

What bad NoSQL looks like...



Much better picture...



Getting the most out of DynamoDB throughput

DynamoDB Developer Guide

"To get the most out of DynamoDB throughput, create tables where the hash key element has a large number of distinct values, and values are requested fairly uniformly, as randomly as possible."

- Space: access is evenly spread over the key-space.
- Time: requests arrive evenly spaced in time.

Connecting to AWS Services

- When interacting with AWS we need to specify our AWS security credentials
  - Verify who we are (Authentication)
  - Verify that we have permission to access the resources (Authorization)
- Alternative: Access services via an EC2 instance that has the appropriate Roles.
  - No need for further authentication/authorization.
  - No security credentials used.
  - We need to access the EC2 instance via our Private Key.

AWS Security Credentials

- AWS uses security credentials to authenticate and authorize your requests.
- Two different types of users in AWS:
  - The account owner (root user) – created automatically,
  - An AWS Identity and Access Management (IAM) user – created manually.
- All AWS users have security credentials.

AWS Users

- Root user
  - The credentials of the account owner allow full access to all resources in the account.
  - You cannot explicitly deny the root user access to resources.
  - For this reason it is highly recommended:
    - Create an IAM User with administration permissions.
    - Stop using Root user.
  - There are specific tasks that are restricted to the AWS account root user – e.g., close the account.
- IAM User
  - Securely control access to AWS services and resources for users in your AWS account.
  - Grant/Revoke policies on the fly.

AWS credentials for programmatic access

- You must provide your AWS access keys to make programmatic calls to AWS.
  - Access Key + Secret Key + Session Token.
- The secret access key is available for download only when you create it.
- If you don't download your secret access key or if you lose it, you must create a new one.
- You can assign up to two access keys per user (root user or IAM user).
- You can disable a key – but it counts toward your limit of two access keys.
- After you delete an access key, it's gone forever and can't be restored, but it can be replaced with a new access key.

How to create an access key

1. Sign in to the AWS Management Console as the root user.
2. In the navigation bar on the upper right, choose your account name or number and then choose My Security Credentials.
3. Expand the Access keys (access key ID and secret access key) section.
4. Choose Create New Access Key.
   - If you already have two access keys, this button is disabled.
5. When prompted, choose Show Access Key or Download Key File. This is your only opportunity to save your secret access key.
6. After you've saved your secret access key in a secure location, chose Close.

AWS CLI

- You can provide your AWS access keys to AWS CLI

```
# aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: ENTER
```

AWS Python SDK (boto3)

```
pip3 install boto3
```

- Boto3 has two distinct levels of APIs.
  1. Client (or "low-level") APIs provide one-to-one mappings to the underlying HTTP API operations.
  2. Resource APIs hide explicit network calls but instead provide resource objects and collections to access attributes and perform actions.

```
ec2 = boto3.client('ec2', 'eu-central-1')
response = ec2.describe_instances()
print(response)

for i in ec2.instances.all():
    if i.state['Name'] == 'stopped':
        i.start()
```

AWS Python SDK (boto3)

```
pip3 install boto3
```

- Boto3 has two distinct levels of APIs.
  1. Client (or "low-level") APIs provide one-to-one mappings to the underlying HTTP API operations.
  2. Resource APIs hide explicit network calls but instead provide resource objects and collections to access attributes and perform actions.

```
ec2 = boto3.client('ec2', 'eu-central-1')
response = ec2.describe_instances()
print(response)

for i in ec2.instances.all():
    if i.state['Name'] == 'stopped':
        i.start()
```

Specifying Access Keys to boto3

- If not specified, boto3 uses default Access Key.

```
# Get resources from the default session
s3 = boto3.resource('s3')
```

- It is possible to specify upon connecting to a resource.

```
# Get resources from the default session
s3 = boto3.resource('s3',
                    aws_access_key_id=ACCESS_KEY,
                    aws_secret_access_key=SECRET_KEY,
                    aws_session_token=SESSION_TOKEN)
```

Connecting to an existing DynamoDB Table

```
# Connect to dynamodb resources
ddb = boto3.resource('dynamodb',region_name='eu-central-1')

# Connect to specific table
table = dynamodb.Table('Books')

print(table.creation_date_time)
print(table.item_count)
print(table.key_schema)
print(table.provisioned_throughput)
```

Inserting a record to a DynamoDB Table

```
entry = { "Author": "Orson Scott Card",
          "Book Title" : "Ender's Game",
          "Language" : "English",
          "Publication Date" : "15/01/1985",
          "ISBN" : "9780812550702"}

# Connect to dynamodb resources
ddb = boto3.resource('dynamodb',region_name='eu-central-1')

# Connect to specific table
table = ddb.Table('Books')

# Store new entry
table.put_item(Item=entry)
```

Inserting a record to a DynamoDB Table (2)

```
entry = { "Author": "Frank Herbert",
          "Book Title" : "Dune",
          "Language" : "English",
          "Publication Date" : "01/06/1965",
          "ISBN" : "9780593099322",
          "Characters" : ["Stilgar", "Vladimir Harkonnen",
              "Duncan Idaho", "Leto Atreides", "Paul Atreides",
              "Alia Atreides", "Lady Jessica", "Shaddam IV",
              "Gurney Halleck"],
          "Literary Awards": ["Hugo Award for Best Novel (1966)",
              "Nebula Award for Best Novel (1965)",
              "Seiun Award for Best Foreign Novel (1974)"]}

# Store new entry
table.put_item(Item=entry)
```

Inserting multiple records to a DynamoDB Table (2)

- Speed up the process + reduce the number of write requests.
- Automatically handles buffering and sending items in batches.
- Automatically handles any unprocessed items and resend them as needed.

```
with table.batch_writer() as batch:
    batch.put_item(Item={'Author': 'Frank Herbert',
                         'Book Title': 'Dune Messiah'})
    batch.put_item(Item={'Author': 'Frank Herbert',
                         'Book Title': 'Children of Dune',
                         'Publication Date': '21/04/1976'})
    batch.put_item(Item={'Author': 'Frank Herbert',
                         'Book Title': 'The Great Dune Trilogy',
                        'ISBN': '9780575070707'})
```

Getting a record from a DynamoDB Table

```
my_key = { "Author": "Orson Scott Card",
           "Book Title" : "Ender's Game"}

# Connect to dynamodb resources
ddb = boto3.resource('dynamodb',region_name='eu-central-1')

# Connect to specific table
table = dynamodb.Table('Books')

# Store new entry
response = table.get_item(Key=my_key)
print(response)
```

Updating a record from a DynamoDB Table

```
my_key = { "Author": "Ray Bradbury",
           "Book Title": "Fahrenheit 451"}

# Update entry
response = table.update_item(Key=my_key,
        UpdateExpression="set ISBN=:i",
        ExpressionAttributeValues={
            ':i': 9780743247221
        },
        ReturnValues="UPDATED_NEW"
    )
print(response)
```

Updating a record from a DynamoDB Table (2)

```
my_key = { "Author": "Ray Bradbury",
           "Book Title": "Fahrenheit 451"}

# Update entry
response = table.update_item(Key=my_key,
        UpdateExpression="set #la=:l",
        ExpressionAttributeNames={"#la": "Language"},
        ExpressionAttributeValues={
            ':l': "English"
        },
        ReturnValues="UPDATED_NEW"
    )
print(response)
```

Deleting a record from a DynamoDB Table

```
my_key = { "Author": "Ray Bradbury",
           "Book Title": "Fahrenheit 451"}

# Delete entry
response = table.delete_item(Key=my_key)
print(response)
```

Query records from a DynamoDB Table

- You can use the query method to retrieve data from a table.
- You must specify a partition key value.
- The sort key is optional.

```
from boto3.dynamodb.conditions import Key

# Query entries
response = table.query(
       KeyConditionExpression=Key('Author').eq('Frank Herbert')
       )
print(response)
```

Scan records from a DynamoDB Table

- The scan method reads every item in the entire table and returns all the data in the table.
- You can provide an optional FilterExpression so that only the items matching your criteria are returned.
- The filter is applied only after the entire table has been scanned.

```
from boto3.dynamodb.conditions import Attr

# Query entries
response = table.scan(
    FilterExpression=Attr('Language').eq('English')
)
items = response['Items']
print(items)
```

Create a DynamoDB Table

```
table = ddb.create_table(
        TableName='Authors',
        KeySchema=[
            { 'AttributeName': 'name',
              'KeyType': 'HASH'  }, # Partition key
            { 'AttributeName': 'genre',
              'KeyType': 'RANGE' }  # Sort key
        ],
        AttributeDefinitions=[
            { 'AttributeName': 'name',
              'AttributeType': 'S' },
            { 'AttributeName': 'genre',
              'AttributeType': 'S'},
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        })
```

Delete a DynamoDB Table

```
table = ddb.Table('Authors')

# Delete table
table.delete()
```

1:1 relationships or key-values

Use a table or GSI with an alternate partition key
Use GetItem or BatchGetItem API
Example: Given an SSN or license number, get attributes

Users Table	
Partiton key	Attributes
SSN = 123-45-6789	Email = johndoe@nowhere.com, License = TDL25478134
SSN = 987-65-4321	Email = maryfowler@somewhere.com, License = TDL78309234

Users-Email-GSI	
Partition key	Attributes
License = TDL78309234	Email = maryfowler@somewhere.com, SSN = 987-65-4321
License = TDL25478134	Email = johndoe@nowhere.com, SSN = 123-45-6789

1:N relationships or parent-children

Use a table or GSI with partition and sort key
Use Query API

Example:

- Given a device, find all readings between epoch X, Y

Device-measurements		
Partition Key	Sort key	Attributes
DeviceId = 1	epoch = 5513A97C	Temperature = 30, pressure = 90
DeviceId = 1	epoch = 5513A9DB	Temperature = 30, pressure = 90

N:M relationships

Use a table and GSI with partition and sort key elements switched
Use Query API
Example: Given a user, find all games. Or given a game, find all users.

User-Games-Table	
Partition Key	Sort key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Partition Key	Sort key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

Hierarchical Data Structures as Items…

Use composite sort key to define a Hierarchy
Highly selective result sets with sort queries
Index anything, scales to any size

	Primary Key		Attributes						
	ProductID	type							
Items	1	bookID	title	author	genre	publisher	datePublished	ISBN	
			Ringworld	Larry Niven	Science Fiction	Ballantine	Oct-70	0-345-02046-4	
	2	albumID	title	artist	genre	label	studio	relesed	producer
			Dark Side of the Moon	Pink Floyd	Progressive Rock	Harvest	Abbey Road	3/1/73	Pink Floyd
	2	albumID:trackID	title	length	music	vocals			
			Speak to Me	1:30	Mason	Instrumental			
	2	albumID:trackID	title	length	music	vocals			
			Breathe	2:43	Waters, Gilmour, Wright	Gilmour			
	2	albumID:trackID	title	length	music	vocals			
			On the Run	3:30	Gilmour, Waters	Instrumental			
	3	movieID	title	genre	writer	producer			
			Idiocracy	Scifi Comedy	Mike Judge	20th Century Fox			
	3	movieID:actorID	name	character	image				
			Luke Wilson	Joe Bowers	img2.jpg				
	3	movieID:actorID	name	character	image				
			Maya Rudolph	Rita	img3.jpg				
	3	movieID:actorID	name	character	image				
			Dax Shepard	Frito Pendejo	img1.jpg				

… or as Documents (JSON)

JSON data types (M, L, BOOL, NULL)
Document SDKs Available
Indexing only via Streams/Lambda
400KB max item size (limits hierarchical data structure)

	Primary Key	Attributes							
	ProductID								
Items	1	id	title	author	genre	publisher	datePublished	ISBN	
		bookID	Ringworld	Larry Niven	Science Fiction	Ballantine	Oct-70	0-345-02046-4	
	2	id	title	artist	genre	Attributes			
		albumID	Dark Side of the Moon	Pink Floyd	Progressive Rock	{ label:"Harvest", studio: "Abbey Road", published: "3/1/73", producer: "Pink Floyd", tracks: [{title: "Speak to Me", length: "1:30", music: "Mason", vocals: "Instrumental"},{title: "Breathe", length: "2:43", music: "Waters, Gilmour, Wright", vocals: "Gilmour"},{title: "On the Run", length: "3:30", music: "Gilmour, Waters", vocals: "Instrumental"}]}			
	3	id	title	genre	writer	Attributes			
		movieID	Idiocracy	Scifi Comedy	Mike Judge	{ producer: "20th Century Fox", actors: [{ name: "Luke Wilson", dob: "9/21/71", character: "Joe Bowers", image: "img2.jpg"},{ name: "Maya Rudolph", dob: "7/27/72", character: "Rita", image: "img1.jpg"},{ name: "Dax Shepard", dob: "1/2/75", character: "Frito Pendejo", image: "img3.jpg"}]			