

# Under the Covers of DynamoDB

Steffen Krause  
Technology Evangelist  
@AWS\_Aktuell  
skrause@amazon.de



# Overview

---

1. Getting started
2. Data modeling
3. Partitioning
4. Replication & Analytics

1

# Getting started

# Global infrastructure



Deployment & Administration

App Services

Comput  
e

Storage

Databa  
se

Networking

AWS Global Infrastructure

## Regions

*An independent collection of AWS resources in a defined geography*

*A solid foundation for meeting location-dependent privacy and compliance requirements*

## Availability Zones

*Designed as independent failure zones*

*Physically separated within a typical metropolitan region*

## Edge Locations

*To deliver content to end users with lower latency*

*A global network of edge locations*

*Supports global DNS infrastructure (Route53) and CloudFront CDN*

# Database



Deployment & Administration

App Services

Comput  
e

Storage

Databa  
se

Networking

AWS Global Infrastructure

## Relational Database Service (RDS)

*Database-as-a-Service – MySQL, Oracle, MS SQL Server*

*No need to install or manage database instances*

*Scalable and fault tolerant configurations*

## DynamoDB

*Provisioned throughput NoSQL database*

*Fast, predictable performance*

*Fully distributed, fault tolerant architecture*

## SimpleDB

## Redshift

*Data Warehouse Service up to Petabytes*

*Cost effective, fully managed*

*Easy connection to BI solutions*

# DynamoDB is a managed NoSQL database service.

---

Store and retrieve any amount of data.

Serve any level of request traffic.

Without the operational burden.

# Consistent, predictable performance.

---

Single digit millisecond latency.

Backed on solid-state drives.



# Flexible data model.

---

Key/attribute pairs. No schema required.

Easy to create. Easy to adjust.

# Seamless scalability.

---

No table size limits. Unlimited storage.

No downtime.

# Durable.

---

Consistent, disk only writes.

Replication across data centers and availability zones.

Without the operational burden.

Focus on your app.

Two decisions + three clicks  
= ready for use



Level of throughput

Primary keys

Two decisions + three clicks

= ready for use



Level of throughput

Primary keys

**Two decisions** + three clicks

= ready for use



# Provisioned throughput.

---

Reserve IOPS for reads and writes.

Scale up for down at any time.

# Pay per capacity unit.

---

Priced per hour of provisioned throughput.

# Write throughput.

---

Size of item x writes per second

\$0.0065 for 10 write units

# Consistent writes.

---

Atomic increment and decrement.

Optimistic concurrency control: conditional writes.

# Transactions.

---

Item level transactions only.

Puts, updates and deletes are ACID.

Strong or eventual consistency

Read throughput.

Strong or eventual consistency

# Read throughput.

---

Provisioned units = size of item x reads per second

\$0.0065 per hour for 50 units

Strong or eventual consistency

# Read throughput.

---

Provisioned units =  $\frac{\text{size of item} \times \text{reads per second}}{2}$

\$0.0065 per hour for 100 units



Strong or eventual consistency

# Read throughput.

---

Same latency expectations.

Mix and match at 'read time'.

Provisioned throughput is managed by DynamoDB.

Data is partitioned and  
managed by DynamoDB.

# Reserved capacity.

---

Up to 53% for 1 year reservation.

Up to 76% for 3 year reservation.

# Indexed data storage.

---

\$0.25 per GB per month.

Tiered bandwidth pricing:

[aws.amazon.com/dynamodb/pricing](https://aws.amazon.com/dynamodb/pricing)

# Authentication.

---

Session based to minimize latency.

Uses the Amazon Security Token Service.

Handled by AWS SDKs.

Integrates with IAM.

# Monitoring.

---

CloudWatch metrics:  
latency, consumed read and write throughput,  
errors and throttling.

# Libraries, wrappers and mocks.

---

ColdFusion, Django, Erlang, Java, .Net,  
Node.js, Perl, PHP, Python, Ruby

<http://j.mp/dynamodb-libs>



**DEMO**





## Tables

 Filter: 
[Explore Table](#)
[Create Table](#)
[Modify Throughput](#)
[Delete Table](#)
[Purchase Reserved Capacity](#)

 << < 1 to 1 of 1 tables > >>

Name	Status	Hash Key	Range Key	▲ Read Throughput	Write Throughput
TestTable	ACTIVE	id	-	5	5

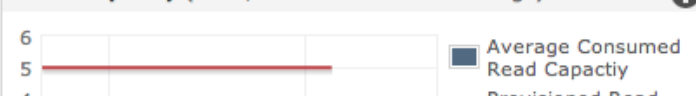
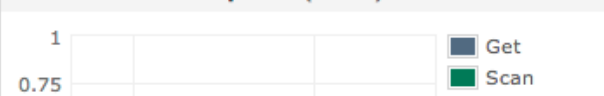
## Table Items

[Details](#)[Indexes](#)**[Monitoring](#)**[Alarm Setup](#)
 ▶ **CloudWatch alarms:** No alarms configured
 [Create Alarm](#)

 ▶ **Alarm History (past 24 hours):** No alarms triggered

**CloudWatch metrics:** Times are displayed in UTC.

 Time Range: 


 Below are your CloudWatch metrics for the selected resources. Click on a graph to see an expanded view. ▶ [View all CloudWatch metrics](#)
**Read Capacity** (Units/Second - 5 Minute Average) 

**Throttled Read Requests** (Count)




## Tables

Filter:

Name

TestTable

## Table Items

Details

▶ CloudWatch alarm

▶ Alarm History (0)

CloudWatch metrics

Below are your CloudWatch metrics

Read Capacity (Units)

6

5

## Create Table

Cancel



PRIMARY KEY

ADD INDEXES  
(optional)PROVISIONED  
THROUGHPUT CAPACITYTHROUGHPUT ALARMS  
(optional)

SUMMARY

**Table Name:**

GameTable

Table will be created in eu-west-1 region

**Primary Key:**

DynamoDB is a schema-less database. You only need to tell us your primary key attribute(s).

Primary Key Type:  Hash and Range  Hash String  Number  Binary

Hash Attribute Name:

Game

 String  Number  Binary

Range Attribute Name:

Date



Choose a hash attribute that ensures that your workload is evenly distributed across hash keys.

For example, "Customer ID" is a good hash key, while "Game ID" would be a bad choice if most of your traffic relates to a few popular games.

[Learn more about choosing your primary key](#)

Cancel

Continue


Help


PRIMARY KEY




ADD INDEXES  
(optional)PROVISIONED  
THROUGHPUT CAPACITYTHROUGHPUT ALARMS  
(optional)

SUMMARY

## Add Indexes (optional)

 Add one or more Local Secondary Indexes 


 You can only define local secondary indexes at table creation time, and cannot remove or modify them later. Local secondary indexes are not appropriate for every application; see [Secondary Indexes](#) and [Item Collections](#).

**Index Hash Key:** Game (String)**Attribute to Index:\***  **Attribute to Index Type:\***  String  Number  Binary**Index Name:\***  **Projected Attributes:\***    

Index Name	Attribute To Index	Projected Attributes
------------	--------------------	----------------------

No Local Secondary Indexes exist. Enter values and click Add Index to create one.

Back

Continue Help 



## Create Table

Cancel

PRIMARY KEY

ADD INDEXES  
(optional)PROVISIONED  
THROUGHPUT CAPACITYTHROUGHPUT ALARMS  
(optional)

SUMMARY

**Provisioned Throughput Capacity:**

- 
- Help me calculate how much throughput capacity I need to provision

**Throughput capacity to provision:**

Amazon DynamoDB lets you specify how much read and write throughput capacity you wish to provision for your table. Using this information, Amazon will provision the appropriate resources to meet your throughput needs. [More Information](#)

Read Capacity Units: Write Capacity Units: 

Throughput capacity for this table will cost up to \$16.41 per month if you have exceeded the [free tier](#).

If you exceed the free tier you are charged for the provisioned throughput capacity of your table **even if you do not actively use your provisioned capacity**. [Learn more about DynamoDB's free tier and pricing.](#)

Back

Continue

Help



## Tables

Filter: 

## Name

TestTable

## Table Items

Details

In

▶ CloudWatch alarm

▶ Alarm History (

CloudWatch me

Below are your Clo

Read Capacity (Unit

6

5

## Configure Alarms

Cancel 

PRIMARY KEY

ADD INDEXES  
(optional)PROVISIONED  
THROUGHPUT CAPACITYTHROUGHPUT ALARMS  
(optional)

SUMMARY

## Throughput Alarms (optional)

 Use Basic Alarms

Notify me when my table's request rates exceed  of Provisioned Throughput for 60 minutes.

Notification will be sent when:

- Read Capacity Units consumed > 40
- or
- Write Capacity Units consumed > 16

Send notification to:

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service.

Advanced alarm settings are available in the CloudWatch Management Console.

[Back](#)[Continue](#) [Help](#)

2

# Data modeling

id = 100	date = 2012-05-16- 09-00-10	total = 25.00
id = 101	date = 2012-05-15- 15-00-11	total = 35.00
id = 101	date = 2012-05-16- 12-00-10	total = 100.00



# Table

id = 100	date = 2012-05-16- 09-00-10	total = 25.00
id = 101	date = 2012-05-15- 15-00-11	total = 35.00
id = 101	date = 2012-05-16- 12-00-10	total = 100.00

id = 100	date = 2012-05-16- 09-00-10	total = 25.00	Item
id = 101	date = 2012-05-15- 15-00-11	total = 35.00	
id = 101	date = 2012-05-16- 12-00-10	total = 100.00	

id = 100	date = 2012-05-16- 09-00-10	total = 25.00 Attribute
id = 101	date = 2012-05-15- 15-00-11	total = 35.00
id = 101	date = 2012-05-16- 12-00-10	total = 100.00

# Where is the schema?

---

Tables do not require a formal schema.

Items are an arbitrarily sized hash.

# Indexing.

---

Items are indexed by primary and secondary keys.

Primary keys can be composite.

Secondary keys are local to the table.

ID

Date

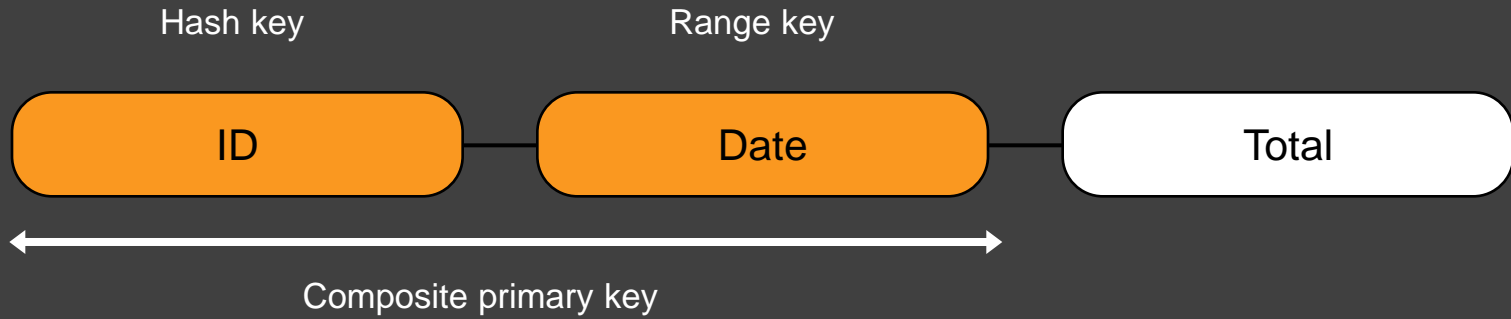
Total

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Hash key



id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00



id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00



Hash key

Range key

Secondary range key



id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

# Programming DynamoDB.

---

Small but perfectly formed API.

CreateTable

PutItem

UpdateTable

GetItem

DeleteTable

UpdateItem

DescribeTable

DeleteItem

ListTables

BatchGetItem

Query

BatchWriteItem

Scan

CreateTable

PutItem

UpdateTable

GetItem

DeleteTable

UpdateItem

DescribeTable

DeleteItem

ListTables

BatchGetItem

Query

BatchWriteItem

Scan

CreateTable

PutItem

UpdateTable

GetItem

DeleteTable

UpdateItem

DescribeTable

DeleteItem

ListTables

BatchGetItem

Query

BatchWriteItem

Scan

```
dynamoDB = new AmazonDynamoDBClient(new ClasspathPropertiesFileCredentialsProvider());
```

```
dynamoDB.setEndpoint("https://dynamodb.eu-west-1.amazonaws.com");
```

```
CreateTableRequest createTableRequest = new CreateTableRequest().withTableName(tableName)
```

```
    .withKeySchema(new KeySchema(new KeySchemaElement().withAttributeName("name").withAttributeType("S")))
```

```
    .withProvisionedThroughput(new ProvisionedThroughput().withReadCapacityUnits(10L).withWriteCapacityUnits(10L));
```

```
TableDescription createdTableDescription = dynamoDB.createTable(createTableRequest).getTableDescription();
```

```
//Wait for table to become available
```

```
DescribeTableRequest describeTableRequest = new DescribeTableRequest().withTableName(tableName);
```

```
TableDescription tableDescription = dynamoDB.describeTable(describeTableRequest).getTable();
```

```
Map<String, AttributeValue> item = newItem("Bill & Ted's Excellent Adventure", 1989, "****", "James", "Sara");
```

```
PutItemRequest putItemRequest = new PutItemRequest(tableName, item);
```

```
PutItemResult putItemResult = dynamoDB.putItem(putItemRequest);
```

```
HashMap<String, Condition> scanFilter = new HashMap<String, Condition>();
```

```
Condition condition = new Condition()
```

```
    .withComparisonOperator(ComparisonOperator.GT.toString())
```

```
    .withAttributeValueList(new AttributeValue().withN("1985"));
```

```
scanFilter.put("year", condition);
```

```
ScanRequest scanRequest = new ScanRequest(tableName).withScanFilter(scanFilter);
```

```
ScanResult scanResult = dynamoDB.scan(scanRequest);
```

# Conditional updates.

---

PutItem, UpdateItem, DeleteItem can take optional conditions for operation.

UpdateItem performs atomic increments.

# One API call, multiple items

---

BatchGet returns multiple items by key.

BatchWrite performs up to 25 put or delete operations.

Throughput is measured by IO, not API calls.



CreateTable

PutItem

UpdateTable

GetItem

DeleteTable

UpdateItem

DescribeTable

DeleteItem

ListTables

BatchGetItem

Query

BatchWriteItem

Scan

# Query vs Scan

---

**Query** for Composite Key queries.

**Scan** for full table scans, exports.

Both support pages and limits.

Maximum response is 1Mb in size.

# Query patterns

---

Retrieve all items by **hash key**.

**Range key** conditions:

==, <, >, >=, <=, begins with, between.

Counts. Top and bottom n values.

Paged responses.

EXAMPLE 1:

Mapping relationships.

# Players

user_id = mza	location = Cambridge	joined = 2011-07-04
user_id = jeffbarr	location = Seattle	joined = 2012-01-20
user_id = werner	location = Worldwide	joined = 2011-05-15

# Players

user_id = mza	location = Cambridge	joined = 2011-07-04
user_id = jeffbarr	location = Seattle	joined = 2012-01-20
user_id = werner	location = Worldwide	joined = 2011-05-15

# Scores

user_id = mza	game = angry-birds	score = 11,000
user_id = mza	game = tetris	score = 1,223,000
user_id = werner	location = bejewelled	score = 55,000

# Players

user_id = mza	location = Cambridge	joined = 2011-07-04
user_id = jeffbarr	location = Seattle	joined = 2012-01-20
user_id = werner	location = Worldwide	joined = 2011-05-15

# Scores

user_id = mza	game = angry-birds	score = 11,000
user_id = mza	game = tetris	score = 1,223,000
user_id = werner	location = bejewelled	score = 55,000

# Leader boards

game = angry-birds	score = 11,000	user_id = mza
game = tetris	score = 1,223,000	user_id = mza
game = tetris	score = 9,000,000	user_id = jeffbarr

# Players

user_id = mza	location = Cambridge	joined = 2011-07-04
user_id = jeffbarr	location = Seattle	joined = 2012-01-20
user_id = werner	location = Worldwide	joined = 2011-05-15

Query for scores  
by user

# Scores

user_id = mza	game = angry-birds	score = 11,000
user_id = mza	game = tetris	score = 1,223,000
user_id = werner	location = bejewelled	score = 55,000

# Leader boards

game = angry-birds	score = 11,000	user_id = mza
game = tetris	score = 1,223,000	user_id = mza
game = tetris	score = 9,000,000	user_id = jeffbarr



# Players

user_id = mza	location = Cambridge	joined = 2011-07-04
user_id = jeffbarr	location = Seattle	joined = 2012-01-20
user_id = werner	location = Worldwide	joined = 2011-05-15

## High scores by game

# Scores

user_id = mza	game = angry-birds	score = 11,000
user_id = mza	game = tetris	score = 1,223,000
user_id = werner	location = bejewelled	score = 55,000

# Leader boards

game = angry-birds	score = 11,000	user_id = mza
game = tetris	score = 1,223,000	user_id = mza
game = tetris	score = 9,000,000	user_id = jeffbarr

EXAMPLE 2:

Storing large items.

# Unlimited storage.

---

Unlimited attributes per item.

Unlimited items per table.

Maximum of 64k per item.

# Split across items.

message_id = 1	part = 1	message = <first 64k>
message_id = 1	part = 2	message = <second 64k>
message_id = 1	part = 3	joined = <third 64k>

# Store a pointer to S3.

message_id = 1	message = <a href="http://s3.amazonaws.com...">http://s3.amazonaws.com...</a>
message_id = 2	message = <a href="http://s3.amazonaws.com...">http://s3.amazonaws.com...</a>
message_id = 3	message = <a href="http://s3.amazonaws.com...">http://s3.amazonaws.com...</a>

EXAMPLE 3:

Time series data

# Hot and cold tables.

April

event_id = 1000	timestamp = 2013-04-16-09-59-01	key = value
event_id = 1001	timestamp = 2013-04-16-09-59-02	key = value
event_id = 1002	timestamp = 2013-04-16-09-59-02	key = value

March

event_id = 1000	timestamp = 2013-03-01-09-59-01	key = value
event_id = 1001	timestamp = 2013-03-01-09-59-02	key = value
event_id =	timestamp =	key =



December

January

February

March

April



# Archive data.

---

Move old data to S3: lower cost.

Still available for analytics.

Run queries across hot and cold data  
with Elastic MapReduce.

3

# Partitioning

# Uniform workload.

---

Data stored across multiple partitions.

Data is primarily distributed by primary key.

Provisioned throughput is divided evenly across partitions.

To achieve and maintain full provisioned throughput, spread workload evenly across hash keys.

# Non-Uniform workload.

---

Might be throttled, even at high levels of throughput.

BEST PRACTICE 1:

# Distinct values for hash keys.

Hash key elements should have a  
high number of distinct values.

Lots of users with unique user\_id.

Workload well distributed across hash key.

user_id = mza	first_name = Matt	last_name = Wood
user_id = jeffbarr	first_name = Jeff	last_name = Barr
user_id = werner	first_name = Werner	last_name = Vogels
user_id = simone	first_name = Simone	last_name = Brunozzi
...	...	...

BEST PRACTICE 2:

**Avoid limited hash key values.**

Hash key elements should have a  
high number of distinct values.



Small number of status codes.  
Unevenly, non-uniform workload.

status = 200	date = 2012-04-01-00-00-01
status = 404	date = 2012-04-01-00-00-01
status 404	date = 2012-04-01-00-00-01
status = 404	date = 2012-04-01-00-00-01

### BEST PRACTICE 3:

# Model for even distribution.

Access by hash key value should be evenly distributed across the dataset.

Large number of devices.

Small number which are much more popular than others.

Workload unevenly distributed.

mobile_id = 100	access_date = 2012-04-01-00-00-01
mobile_id = 100	access_date = 2012-04-01-00-00-02
mobile_id = 100	access_date = 2012-04-01-00-00-03
mobile_id = 100	access_date = 2012-04-01-00-00-04
...	...

Sample access pattern.

Workload randomized by hash key.

mobile_id = 100.1	access_date = 2012-04-01-00-00-01
mobile_id = 100.2	access_date = 2012-04-01-00-00-02
mobile_id = 100.3	access_date = 2012-04-01-00-00-03
mobile_id = 100.4	access_date = 2012-04-01-00-00-04
...	...

4

# Reporting & Analytics

# Seamless scale.

---

Scalable methods for data processing.

Scalable methods for backup/restore.

# Amazon Elastic MapReduce.

---

Managed Hadoop service for  
data-intensive workflows.

[aws.amazon.com/emr](https://aws.amazon.com/emr)

```
create external table items_db
(id string, votes bigint, views bigint) stored by
'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
tblproperties
("dynamodb.table.name" = "items",
"dynamodb.column.mapping" =
"id:id,votes:votes,views:views");
```



```
select id, likes, views  
from items_db  
order by views desc;
```

# Summary

---

1. Getting started
2. Data modeling
3. Partitioning
4. Replication & Analytics

Free tier.

[aws.amazon.com/dynamodb](https://aws.amazon.com/dynamodb)

---

# Thank you!

[skrause@amazon.de](mailto:skrause@amazon.de)

@AWS\_Aktuell



# Ressourcen

Getting Started Guide:

<http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html>

<http://aws.amazon.com/de>

Beginnen Sie mit dem Free Tier:

<http://aws.amazon.com/de/free/>

Twitter: @AWS\_Aktuell

Facebook: <http://www.facebook.com/awsaktuell>

Webinare: <http://aws.amazon.com/de/about-aws/events/>

Slides: <http://de.slideshare.net/AWSAktuell/>

Kommen Sie zum AWS Summit am 2. Mai in Berlin

<https://aws.amazon.com/de/aws-summit-2013/berlin/>

