AWS DynamoDB

Amod Kadam

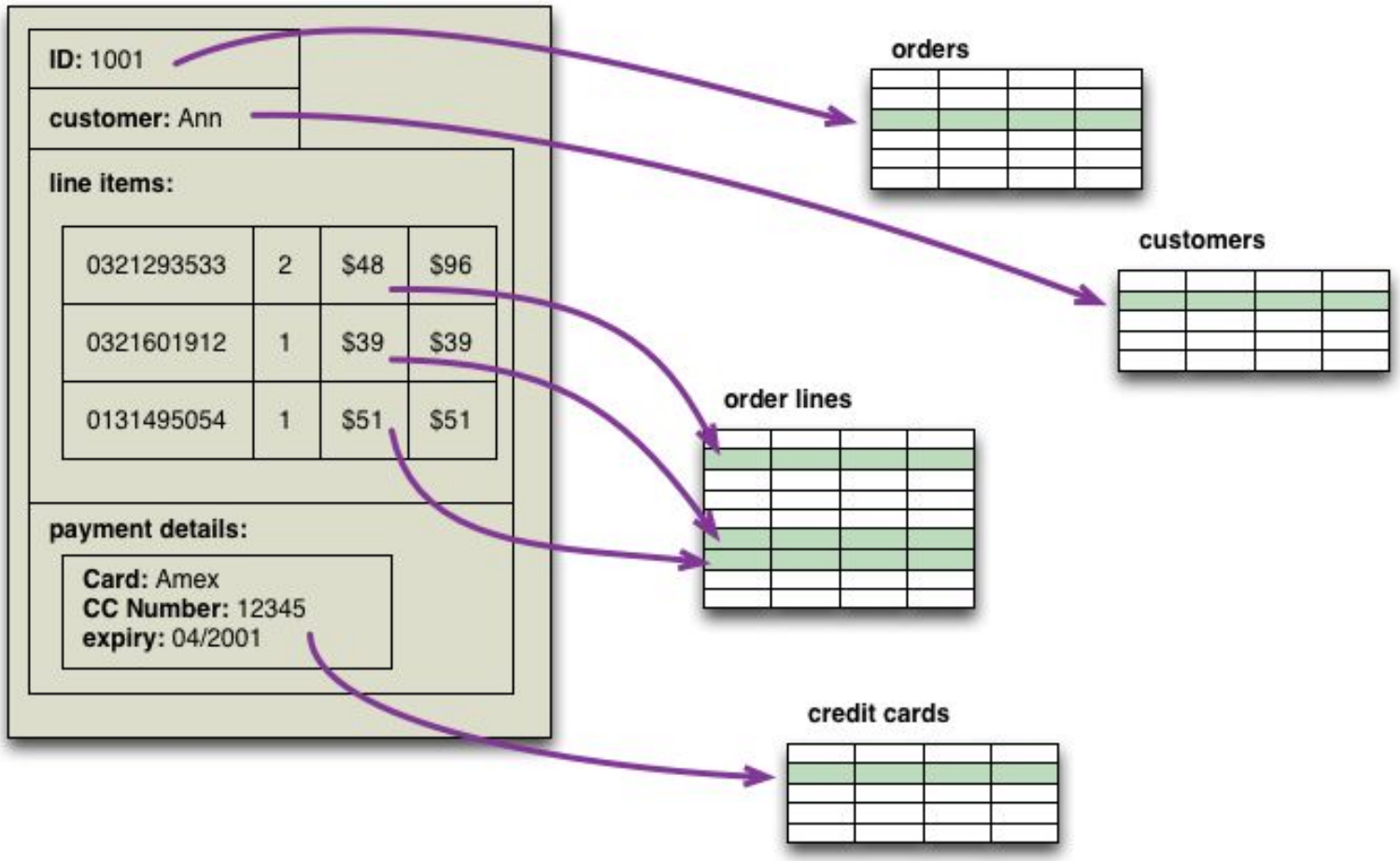
March 2018

First understand NoSQL

Relational Model

- Persistence
- Concurrency
- SQL - Standard
- Transactions
- Integration
- Reporting
- ACID

Data Model


ID: 1001
customer: Ann
line items:
0321293533
2
$48
$96
0321601912
1
$39
$39
0131495054
1
$51
$51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards

Relational Model – Scalability ?

- **ACID**
- Internet of Things (IoT)
- Big Data
- Scaling Up
- Horizontal Scaling
- Sharding
- Replication

Where did this come from ?

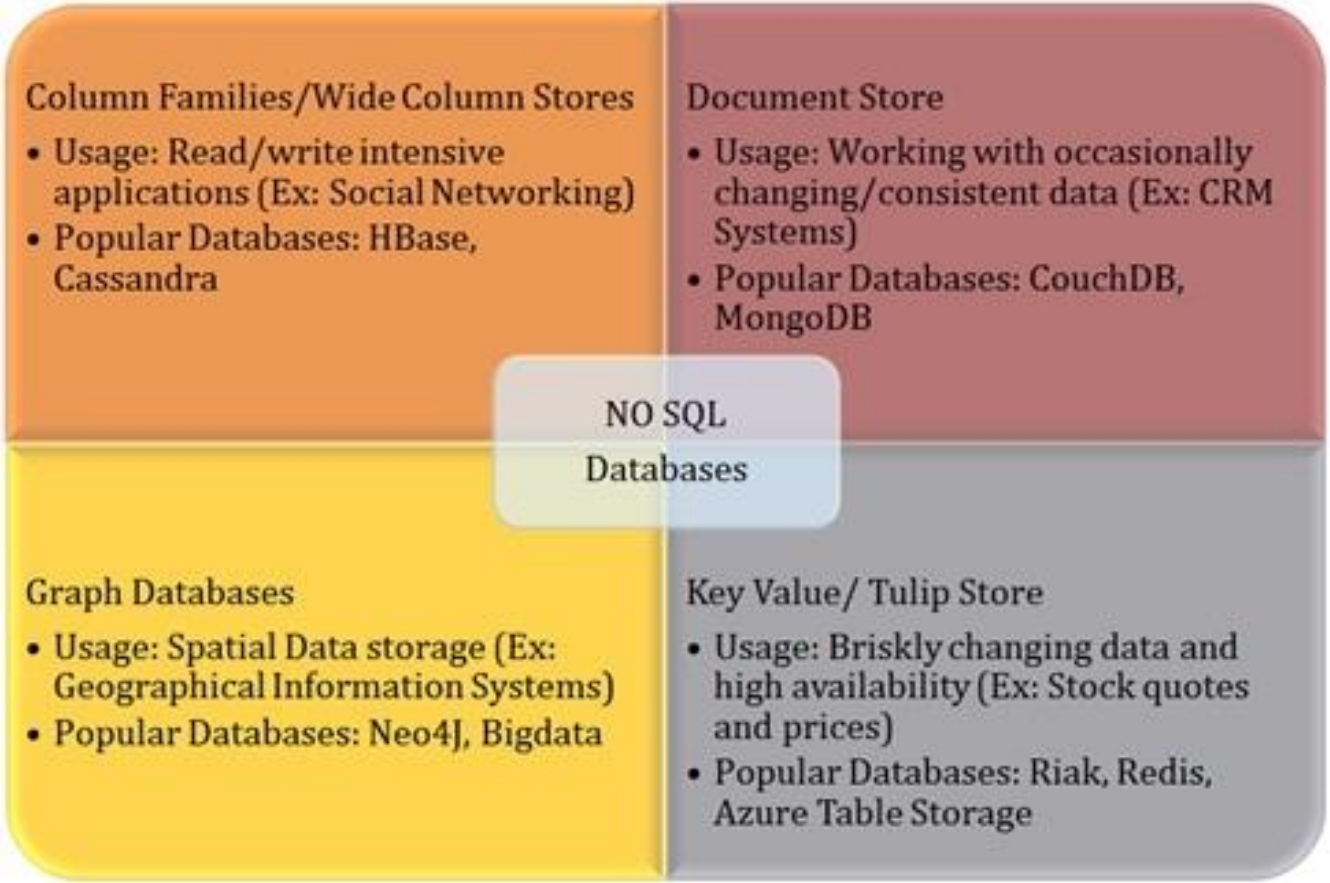
- Meeting to discuss new technologies in the IT market, storage and processing of data
- June 2009
- San Francisco
- Brief term for using as a Twitter hashtag
- **"NoSQL" term was suggested by Eric Evans from RackSpace**
- No deep meaning
- Went viral

Johan Oskarsson

NoSQL Databases

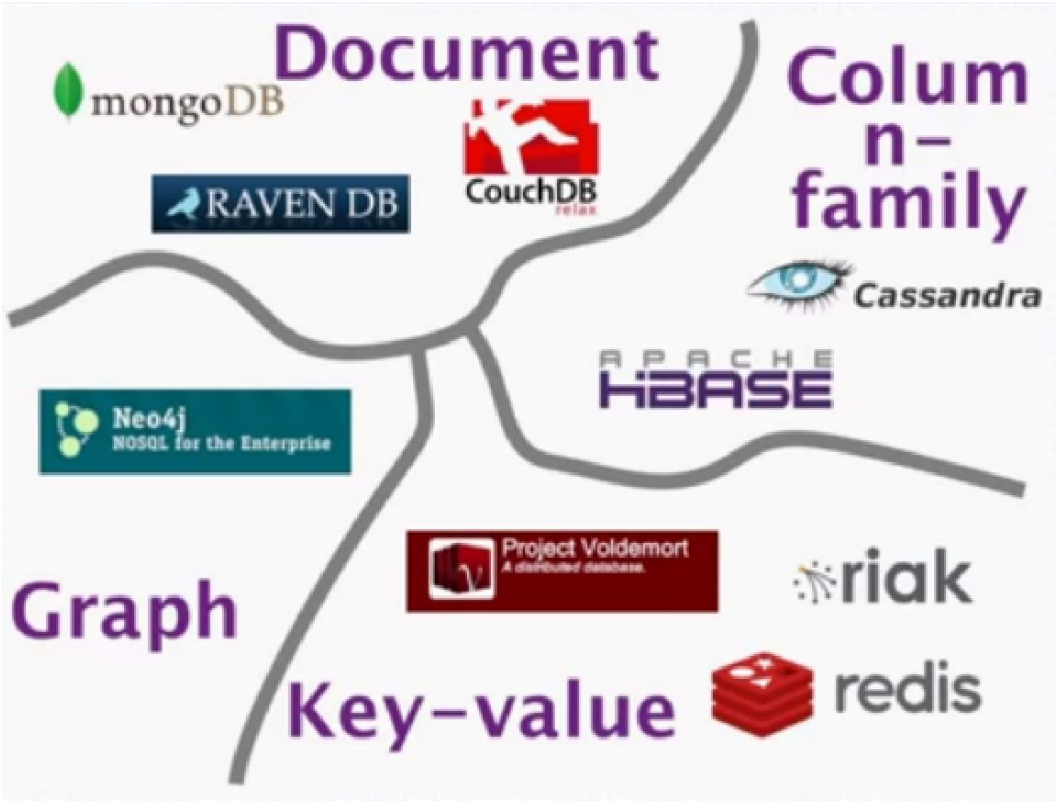
- Not Only SQL
- Non relational data model - **NoREL (No Relational)**
- Simplicity of Design
- Horizontal Scaling
- **Schema less** database (!) – No Fixed Storage Schema
- Faster Response at massive scale
- **No Joins**
- Used in Big Data and Real Time Web Applications

NoSQL – Broad Classification



Source: codeproject.com / Wikipedia

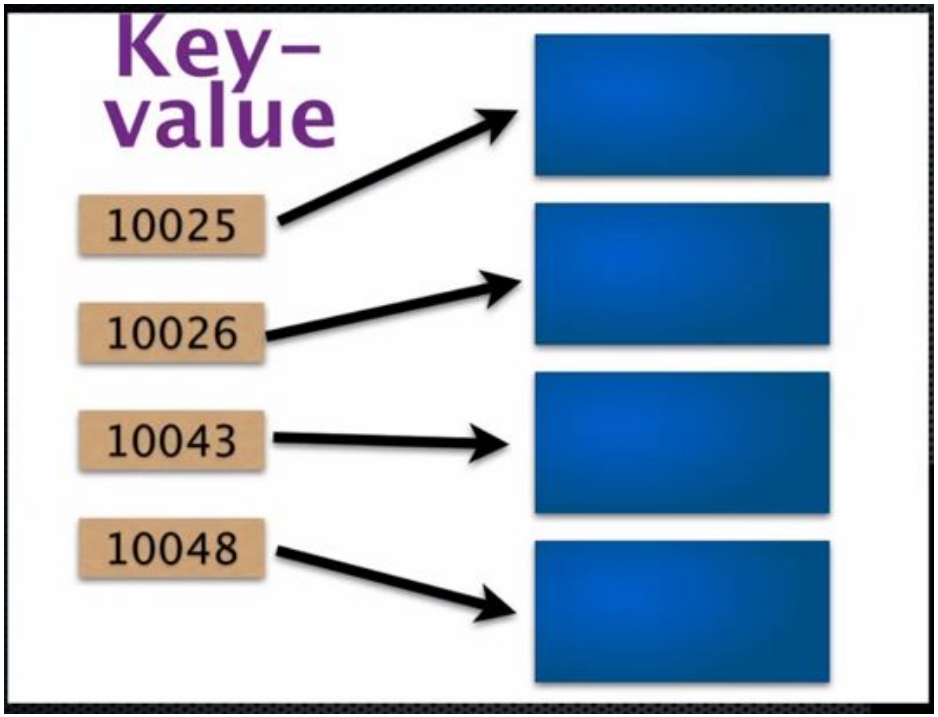
NoSQL Databases - Classification



- **Column**: Accumulo, Cassandra, Druid, HBase
- **Document**: Clusterpoint, Apache CouchDB, Couchbase, MarkLogic, MongoDB
- **Key-value**: Dynamo, FoundationDB, MemcacheDB, Redis, Riak, FairCom c-treeACE
- **Graph**: Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog

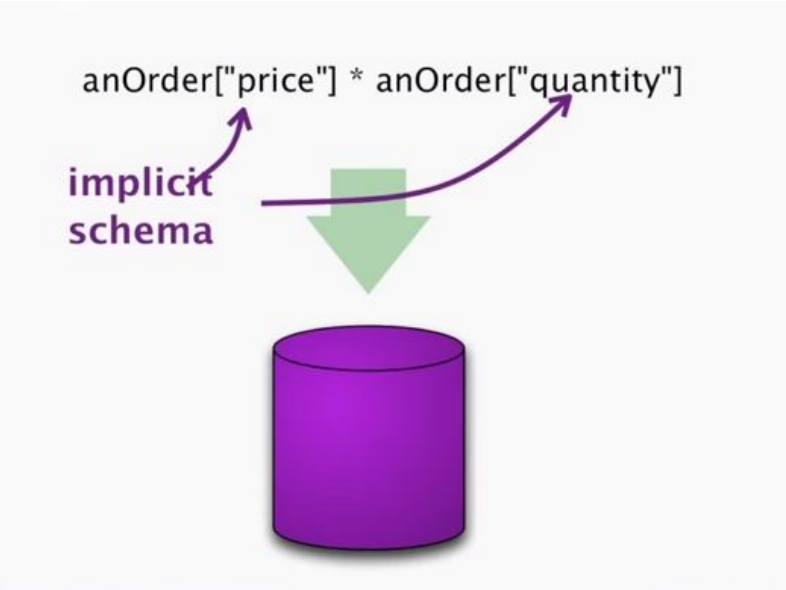
Key Value Store

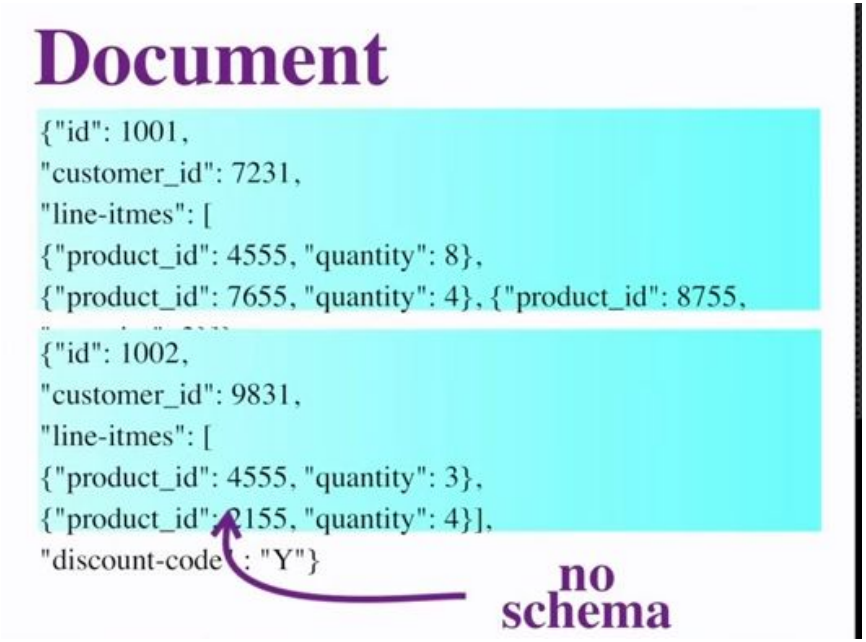
- Distributed Hash table
- Clustered
- Extremely Low Latency
- Files on File System - indexed, massively distributed, handles failover , replicated
- Blobs
- Massive amounts of traffic
- Millions of records/second



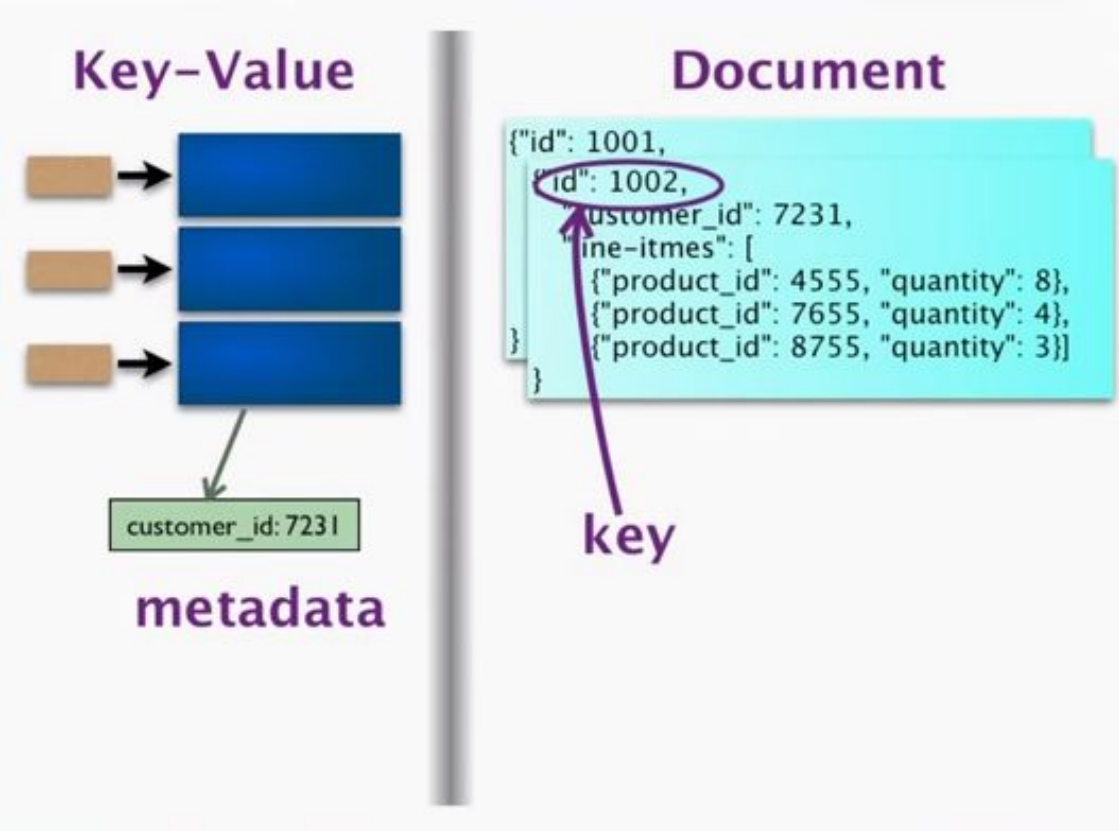
Document Store

- Are also Key Value Store
- Some structured format
- XML / JSON / BSON
- Querying Support

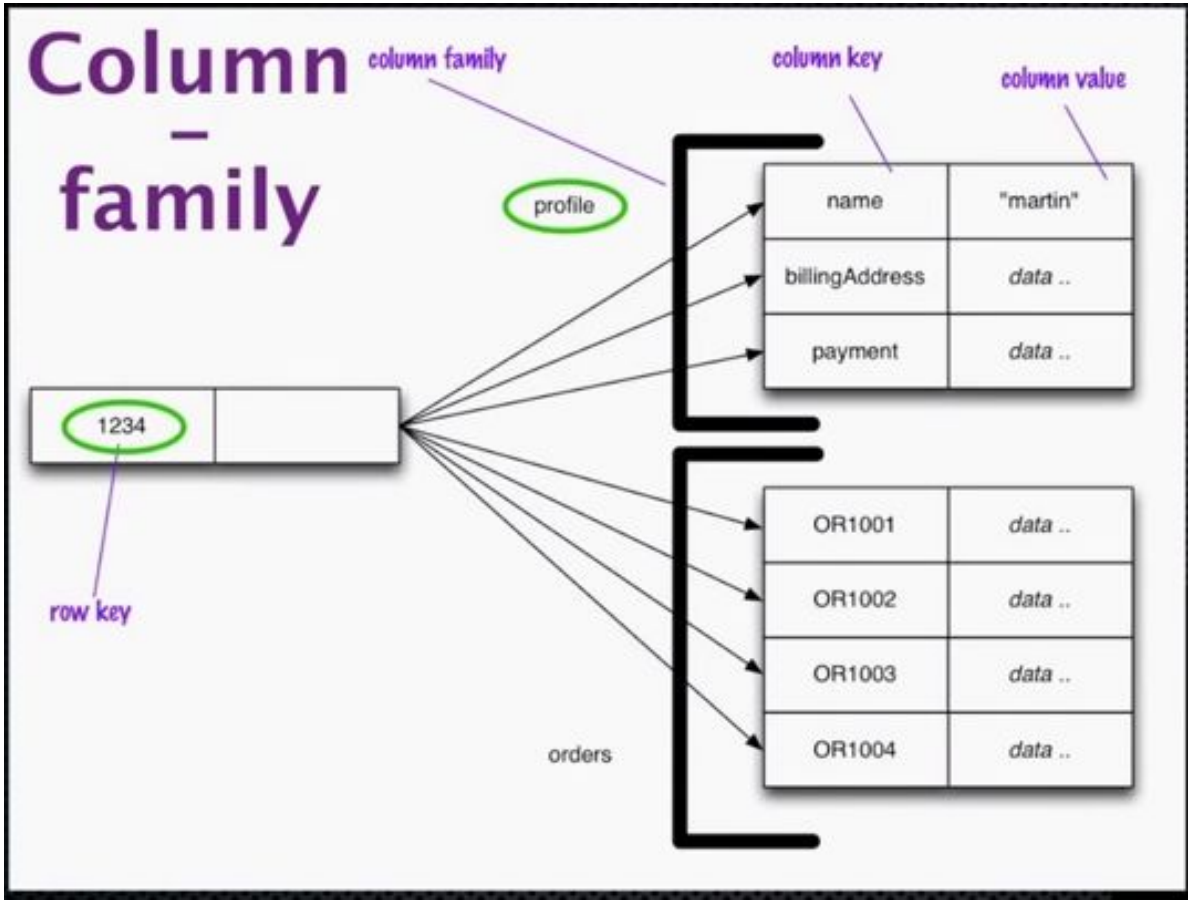




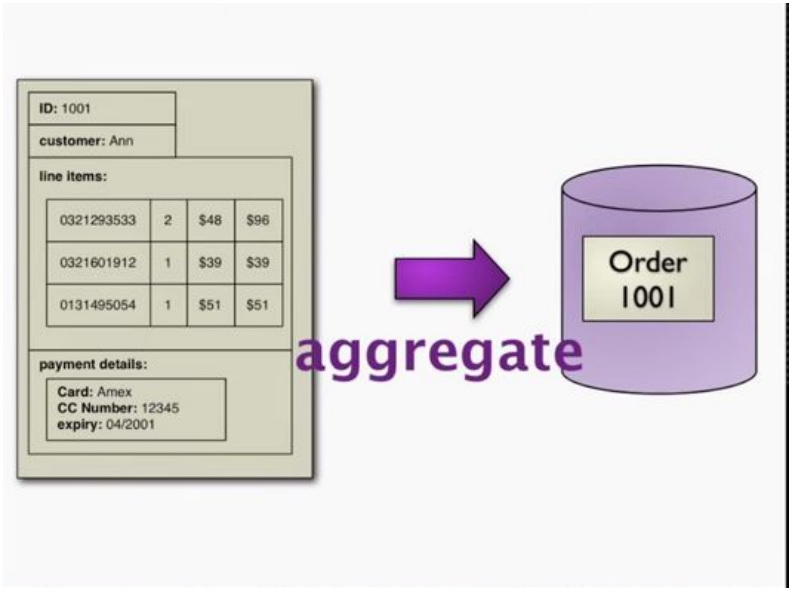
Document Store

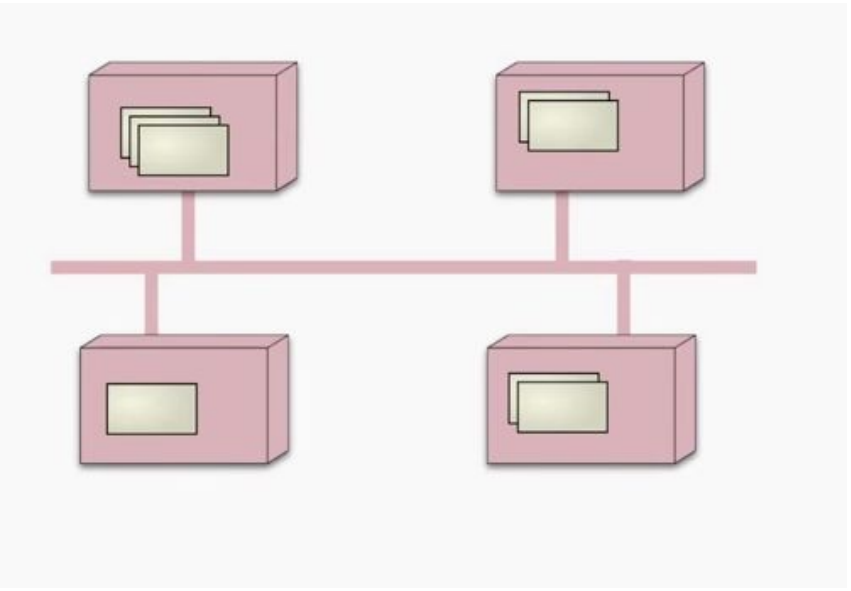


Column Family

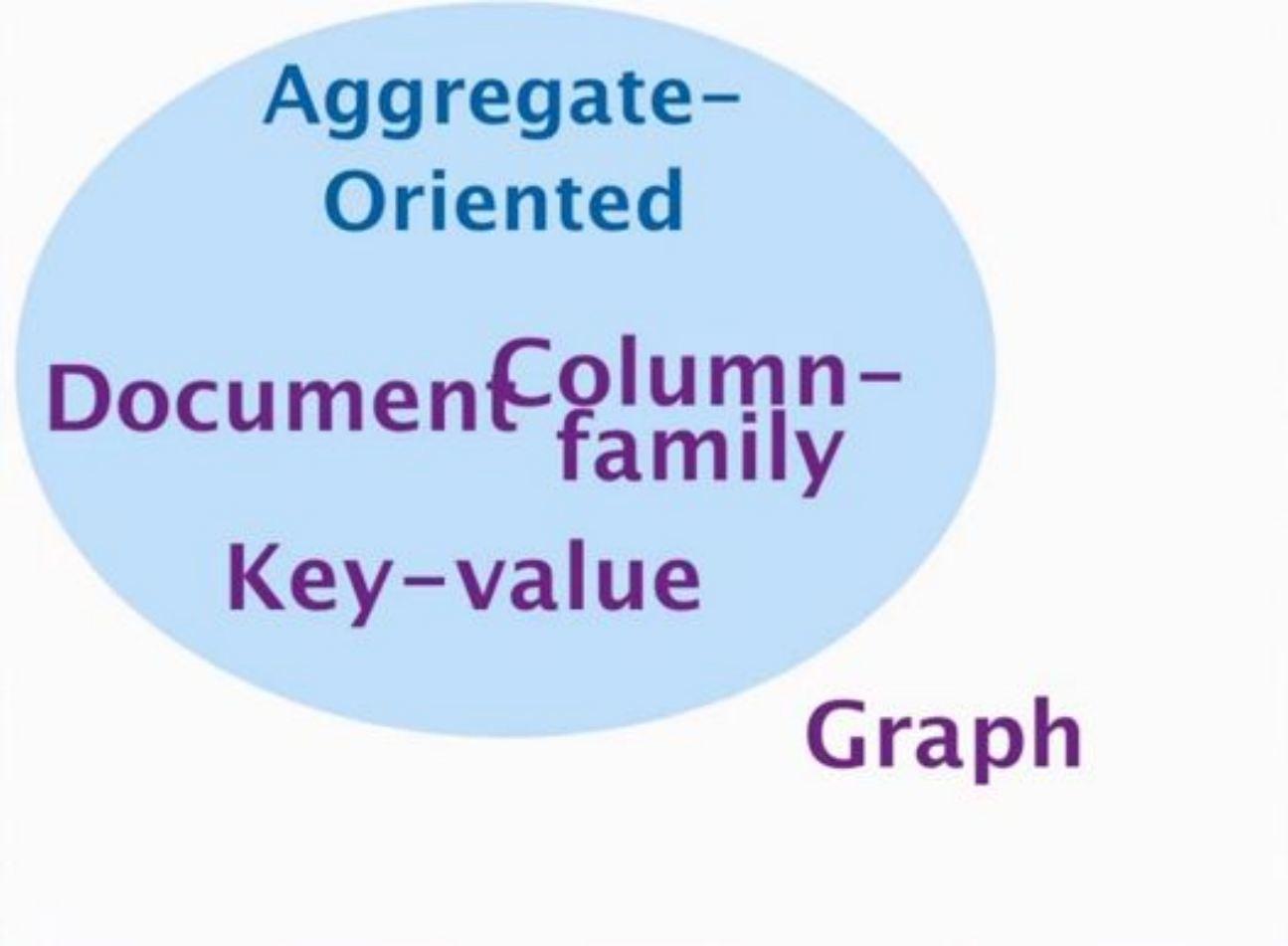


Can we see benefit ?

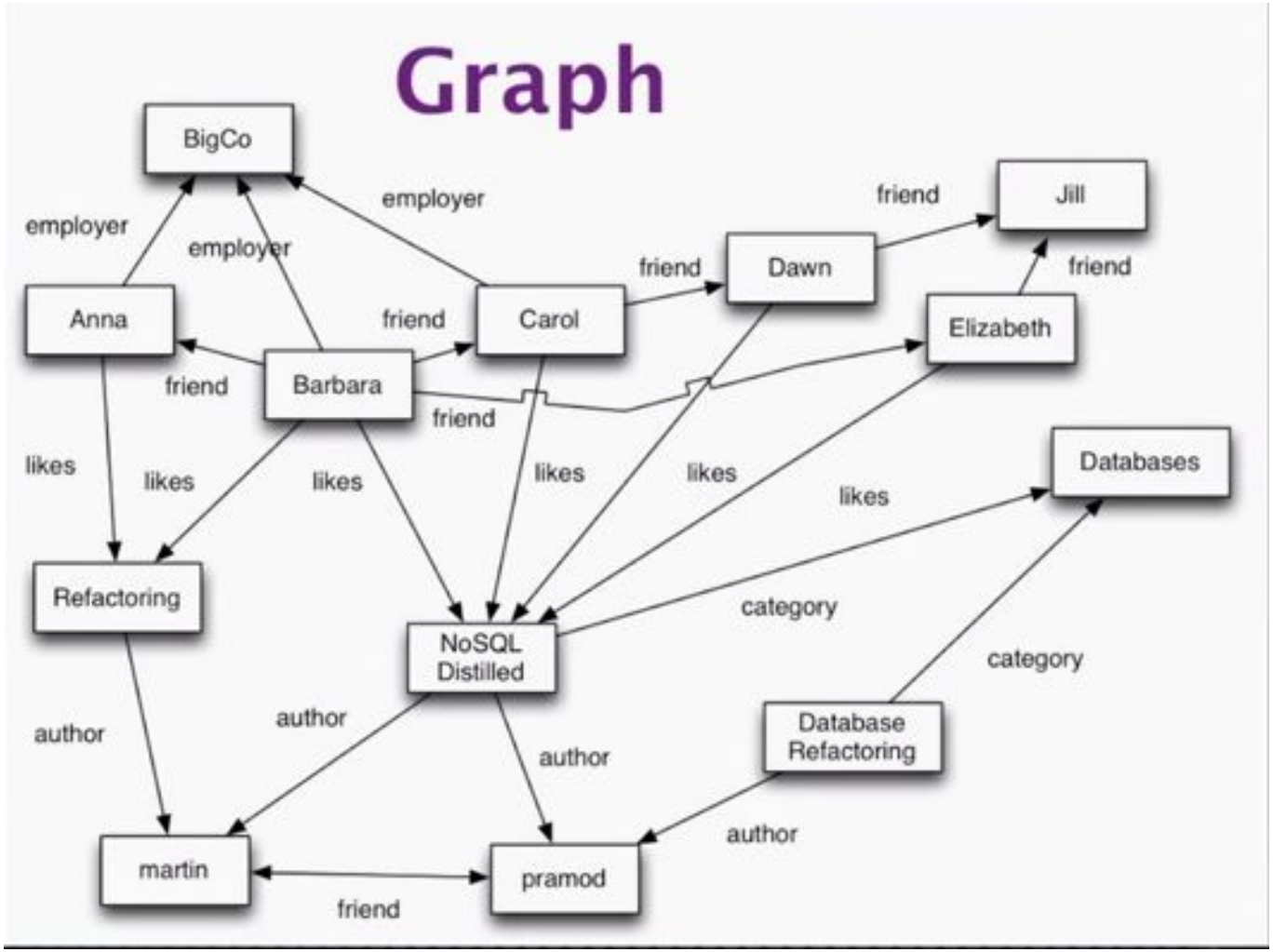




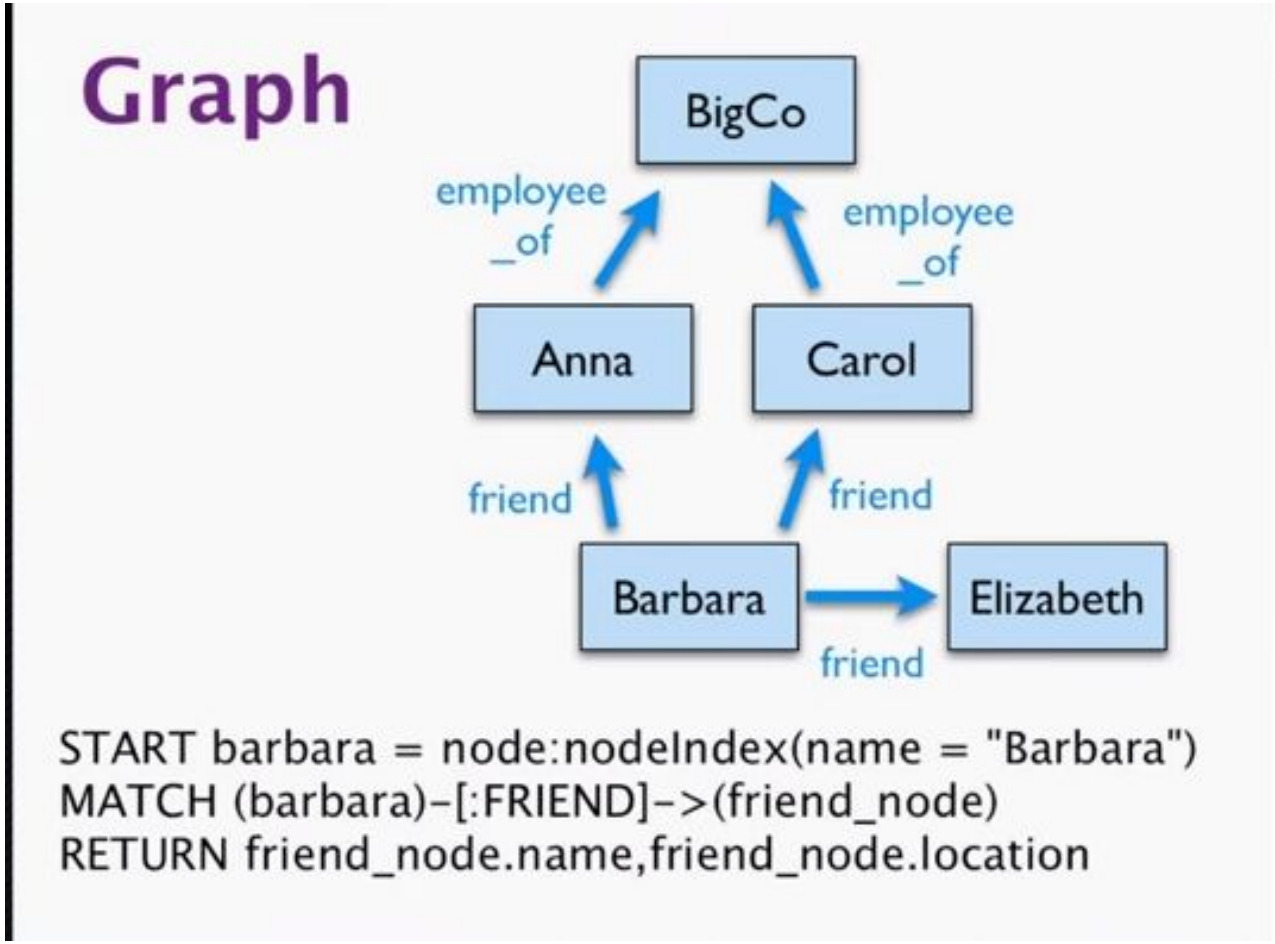
Review of the Classification



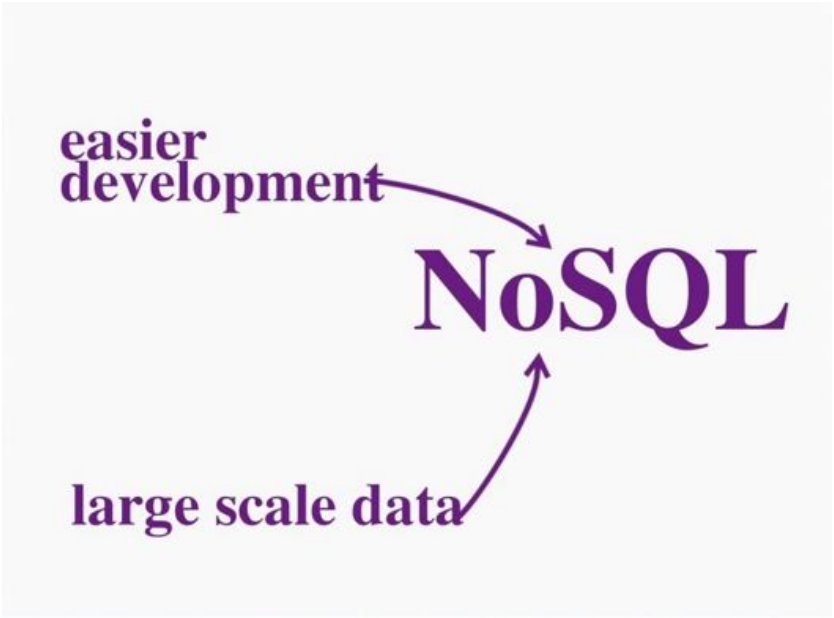
Graph

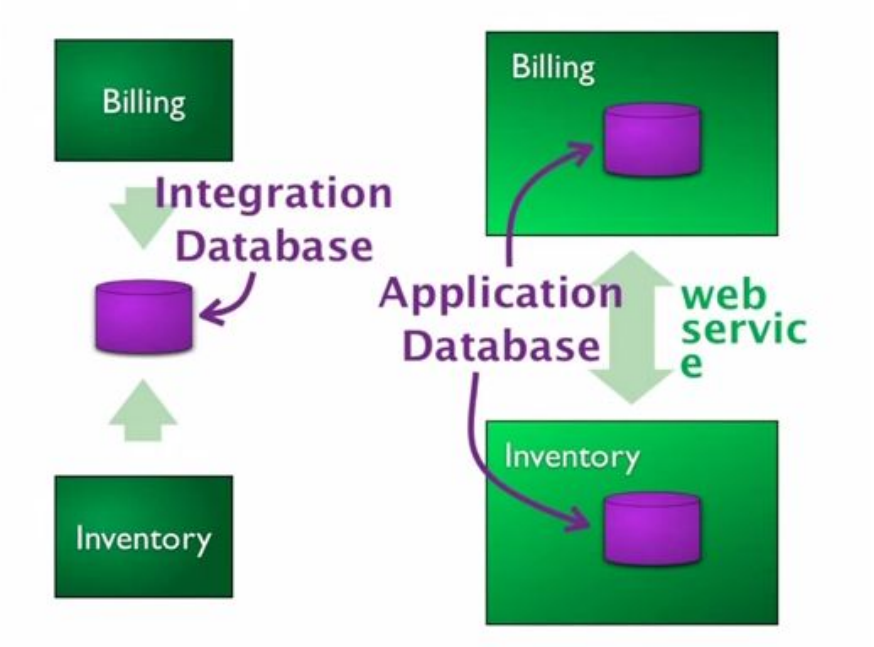

Graph
BigCo
employer
employer
employer
Anna
friend
Carol
friend
Dawn
friend
Jill
friend
Elizabeth
friend
Barbara
friend
friend
likes
likes
likes
likes
likes
likes
likes
Databases
Refactoring
category
NoSQL
Distilled
category
author
author
author
Database
Refactoring
author
martin
pramod
friend

Graph Database Query



How to Choose ?



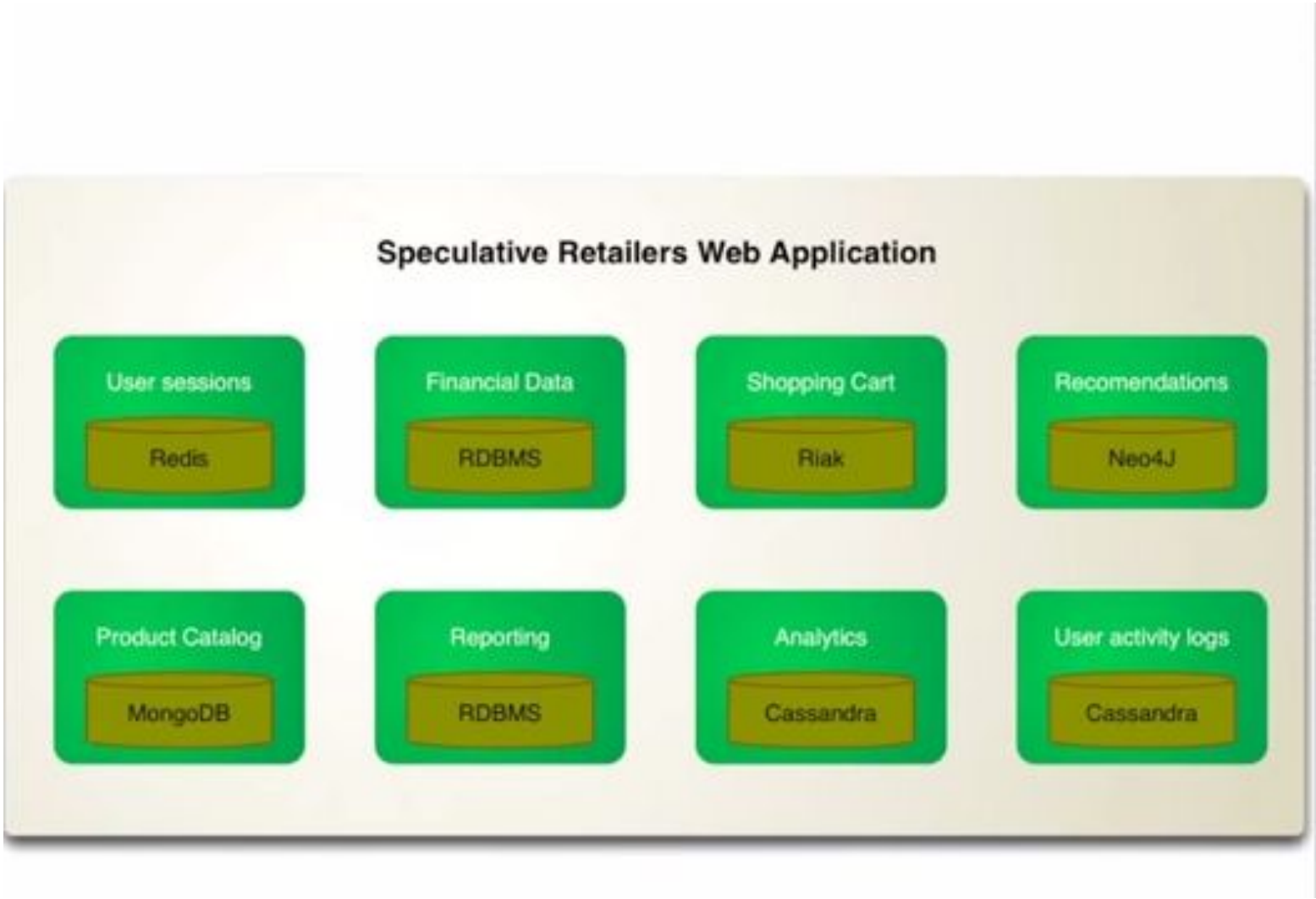


Natural Fit

Articles

Order

Polyglot Persistence



DynamoDB

What is Amazon DynamoDB

- NoSQL database service
- Fully Managed database service
- Supports **Document + Key Value Data Model**
- Does not support cross table joins

Use Cases

- Marketing
- Transactional
  - Order confirmations , Slipping notices , Order Status etc.
- Subscriptions
  - Send newsletters
  - Agendas
- Notifications
- Social Networking
- Business Email Service

Key Concepts

- **Table**
  - Collection of Items
  - Except Primary Key it is schema less

- **Items**
  - Collection of Attributes
  - **Any number** of attributes
  - 400 KB limit on the size

- **Attributes**
  - Single valued
  - Multi valued set - Duplicate Values not allowed

Items

```
ProductCatalog ( Id, ... )
```

```
{
    Id = 101
    ProductName = "Book 101 Title"
    ISBN = "111-1111111111"
    Authors = [ "Author 1", "Author 2" ]
    Price = -2
    Dimensions = "8.5 x 11.0 x 0.5"
    PageCount = 500
    InPublication = 1
    ProductCategory = "Book"
}
```

```
{
    Id = 201
    ProductName = "18-Bicycle 201"
    Description = "201 description"
    BicycleType = "Road"
    Brand = "Brand-Company A"
    Price = 100
    Gender = "M"
    Color = [ "Red", "Black" ]
    ProductCategory = "Bike"
}
```

```
{
    Id = 202
    ProductName = "21-Bicycle 202"
    Description = "202 description"
    BicycleType = "Road"
    Brand = "Brand-Company A"
    Price = 200
    Gender = "M"
    Color = [ "Green", "Black" ]
    ProductCategory = "Bike"
}
```

All product data in same table

Primary Key

- Uniquely identifies the item in DynamoDB
- 2 types of primary keys
- Simple Primary Key /Partition key / Hash Key
  - Primary Key consists of only single attribute
  - Builds an unordered hash index
- **Composite Primary Key /Partition key and sort key / Hash and Range**
  - Primary key consists of two attributes
  - First attribute is hash attribute - unordered hash index
  - Second one is Range Attribute - sorted range index
- Each primary key attribute must be a scalar (meaning that it can hold only a single value).
- Data types allowed for primary key attributes are string, number, or binary

Table Name	Primary Key Type	Hash Attribute Name	Range Attribute Name
Forum (Name, ...)	Hash	Name	-
Thread (ForumName, Subject, ...)	Hash and Range	ForumName	Subject
Reply (Id, ReplyDateTime, ...)	Hash and Range	Id	ReplyDateTime

Read & Write Requirements

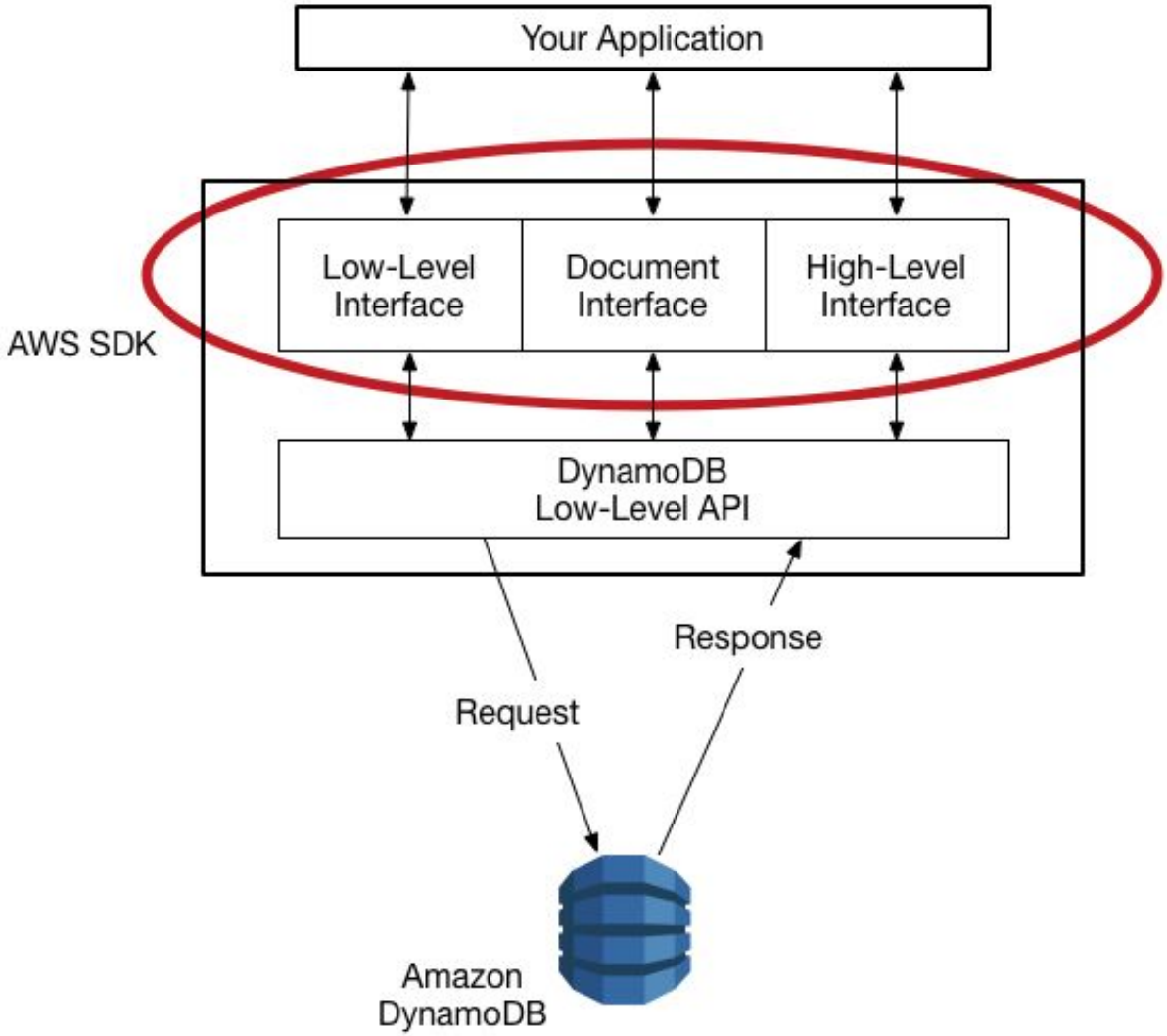
- Ensure High Availability
- Ensure low latency
- Read & Write Throughput
  - Used to reserve sufficient hardware and software resources
  - Appropriately partition your data over multiple **servers**
- Read capacity units
- Write capacity units

Data Types

- Scalar Type
  - Single value
  - string (S) , number, binary , Boolean or null
- Document Type
  - complex ,JSON like
- Set type
  - multiple value , stringset , numberset , binaryset

Programming Interfaces

Programming Interface



1. Low level Interface

- Closely resembles the low-level DynamoDB API request
- Specify data types such as S (String), N (Number) using Data Type Descriptors (DTD)
- Available in every language specific AWS SDK

```
CreateTableRequest request = new CreateTableRequest()
        .withAttributeDefinitions(new AttributeDefinition("CustomerShortName", ScalarAttributeType.S),
                new AttributeDefinition("ForClient", ScalarAttributeType.S))
        .withKeySchema(new KeySchemaElement("CustomerShortName", KeyType.HASH),
                new KeySchemaElement("ForClient", KeyType.RANGE))
        .withProvisionedThroughput(new ProvisionedThroughput(new Long(10), new Long(10)));

request.setTableName("Training");
```

Packages used->

```
com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
com.amazonaws.services.dynamodbv2.model.AttributeAction;
com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
com.amazonaws.services.dynamodbv2.model.AttributeValue;
com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
com.amazonaws.services.dynamodbv2.model.GetItemRequest;
com.amazonaws.services.dynamodbv2.model.GetItemResult;
com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
com.amazonaws.services.dynamodbv2.model.KeyType;
com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
com.amazonaws.services.dynamodbv2.model.PutItemRequest;
com.amazonaws.services.dynamodbv2.model.PutItemResult;
com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
com.amazonaws.services.dynamodbv2.model.UpdateItemRequest;
com.amazonaws.services.dynamodbv2.model.UpdateItemResult;
```

2. Document Interface

- Perform data plane operation (CRUD)
- No need to specify Data Type Descriptor (DTD)
- Data types are implied by Semantics
- Provides methods to convert JSON documents to/from DynamoDB data type
- Supported in Java/.NET/Node.js & JavaScript

3. High Level Interface/Object Persistence/DynamoDBMapper

- Do not perform data plane operations
- Create objects which represent items in DynamoDB
- Write **Object Centric code** rather than database centric code
- Supported in Java and .NET
- The DynamoDBMapper class does not allow you to create, update, or delete tables.
  - for this you have to use low-level-interface

  However you can use `AbstractDynamoDBMapper.generateCreateTableRequest(Class)` to create tables
  http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/dynamodbv2/datamodeling/DynamoDBMapper.html

- Intuitive
- Closer to OO world

Using Interfaces

1. Obtain the service client using client builder

```
AmazonDynamoDB client   client = AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-east-1")).build();
```

2. Use the client interface to invoke operation osn DynamoDB

```
client.createTable(request);
```

Following packages are used

```
com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIgnore;
com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
```

Scan

- Returns one or more items
- Using `FilterExpression` you can have fewer items returned
- If Scanned Item exceeds 1 MB size
  - scan stops
  - result includes `#of items exceeding the limit` and `lastEvaluatedKey`
- Proceeds sequentially by default
- You can request a parallel scan for faster performance operation
- Uses eventually consistent model by default

Query...

- Finds items based on Primary Key values
  - Simple
  - Composite (Hash Key and Sort Key)

- Use a comparison operator to refine the search result
  - For Hash Key EQ
  - For Range/Sort Keys
  - = ,< , <= , > , >= ,BETWEEN etc
- Filter Expression can be used to filter the results
  - A *filter expression* determines which items within the Query results should be returned to you
  - A filter expression is applied after a Query finishes, but before the results are returned. Therefore, a Query will consume the same amount of **read capacity**, regardless of whether a filter expression is present
  - A Query operation can retrieve a maximum of 1 MB of data. This limit applies before the filter expression is evaluated.

Query……

- A filter expression cannot contain partition key or sort key attributes. You need to specify those attributes in the key condition expression, not the filter expression.
- **Paginating the results**
  - http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html#Query.Pagination
- **Capacity Units Consumed**

If you `Query` a...	DynamoDB consumes read capacity units from...
Table	The table's provisioned read capacity.
Global secondary index	The index's provisioned read capacity.
Local secondary index	The base table's provisioned read capacity.

- **Read Consistency**
  - Default mode is EVENTUAL CONSISTENCY
  - means that the Query results might not reflect changes due to recently completed `PutItem` or `UpdateItemoperations`

Working with Tables

Table Operations

`create-table`

- table names must be **unique within region**
- Asynchronous operation
- Upon receiving a **create-table** request, DynamoDB immediately returns a response with a TableStatus of CREATING
- After the table is created, DynamoDB sets the TableStatus to ACTIVE
- You can perform read and write operations only on an ACTIVE table

Working with Items

Performing CRUD Operations

- Create - PutItem
- Read - GetItem
- Update - UpdateItem
- Delete - DeleteItem

UpdateItem

- If an item with the specified key does not exist, UpdateItem creates a new item. Otherwise, it modifies an existing item's attributes.
- *upsert*
- Use `ReturnValues` parameters in request if you want before and after values.

```
updateItemSpec = new UpdateItemSpec().withPrimaryKey( hashKeyName: "year", year, range
        .withUpdateExpression("set info.music_by = :r")
        .withValueMap(new ValueMap().withString( key: ":r", val: "Shankar-Ehsaan-Loy")
        .withReturnValues(ReturnValue.UPDATED_NEW);
```

- Valid values are
  - NONE (default)
  - ALL_OLD
  - UPDATEd_OLD
  - ALL_NEW
  - UPDATED_NEW

ProjectionExpression

- By default DynamoDB returns all attributes
- To restrict the attributes use **ProjectionExpression**
- *projection expression* is a string that identifies the attributes you want
- To retrieve a single attribute, specify its name. For multiple attributes, the names must be comma-separated.

Working with Items

- **batch-write-item**
  - puts or deletes multiple items in one or more tables
  - cannot update items
  - A single call to batch-write-item can write up to 16 MB of data, which can comprise as many as 25 put or delete requests. Individual items to be written can be as large as 400 KB

...

batch-write-item - atomicity ?

- Individual operation is *atomic* but not the entire batch operation
- If any requested operations fail because the table's provisioned throughput is exceeded or an internal processing failure occurs, the failed operations are returned in the **UnprocessedItems** response parameter.
- In order to improve performance with these large-scale operations, batch-write-item does not behave in the same way as individual put-item and delete-item calls would. For example, you cannot specify conditions on individual put and delete requests, and batch-write-item does not return deleted items in the response.

http://docs.aws.amazon.com/cli/latest/reference/dynamodb/batch-write-item.html

Pagination...

- DynamoDB *paginates* the results from Query operations.
- With pagination, the Query results are divided into "pages" of data that are 1 MB in size (or less).
- A single Query will only return a result set that fits within the 1 MB size limit. To determine whether there are more results, and to retrieve them one page at a time, applications should do the following:

...Pagination

1. Examine the low-level Query result:
   - If the result contains a LastEvaluatedKey element, proceed to step 2.
   - If there is *not* a LastEvaluatedKey in the result, then there are no more items to be retrieved.
2. Construct a new Query request, with the same parameters as the previous one—but this time, take the LastEvaluatedKey value from step 1 and use it as the ExclusiveStartKey parameter in the new Queryrequest.
3. Run the new Query request.
4. Go to step 1.

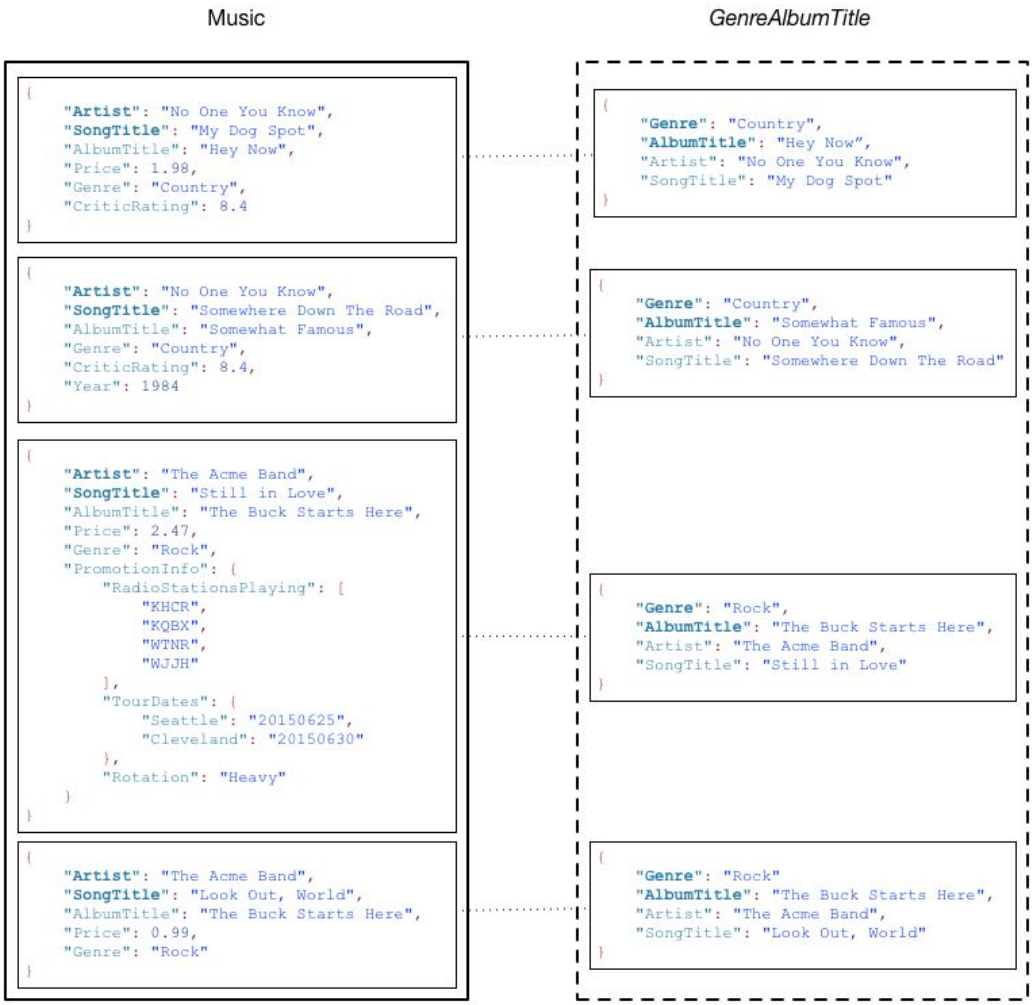
Working with Indexes

Index Types

- Secondary Indexes (alternate key)
  - if you want to query on non-primary attributes secondary indexes are required
- Global Secondary Index (GSI)
  - An index with a partition key and sort key that can be different from those on the table
  - 5 GSI can be created

- Local Secondary Index (LSI)
  - An index that has the same partition key as the table, but a different sort key
  - 5 LSI can be created
- Whenever an Index is created the Primary Key attributes are also copied to index
- One can also ‘project’ or select additional attributes to be put in index

Secondary Index Storage



https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/images/HowItWorksGenreAlbumTitle.png

Comparing GSI and LSI

Parameter	GSI	LSI
Creation time	Can be created any time	Must be created while table creation
Can we modify index after creation	Yes - Only Provisioned Throughput	No ?
Can we delete index after creation	Yes	No ?
…	…	…

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html

Streams

How to handle such use cases ?

- Creating Replica in another region
  - An application in one AWS region modifies the data in a DynamoDB table.
  - A second application in another AWS region reads these data modifications and writes the data to another table, creating a replica that stays in sync with the original table.
- Real Time Metrics
  - popular mobile app modifies data in a DynamoDB table, at the rate of thousands of updates per second.
  - Another application captures and stores data about these updates, providing near real time usage metrics for the mobile app.

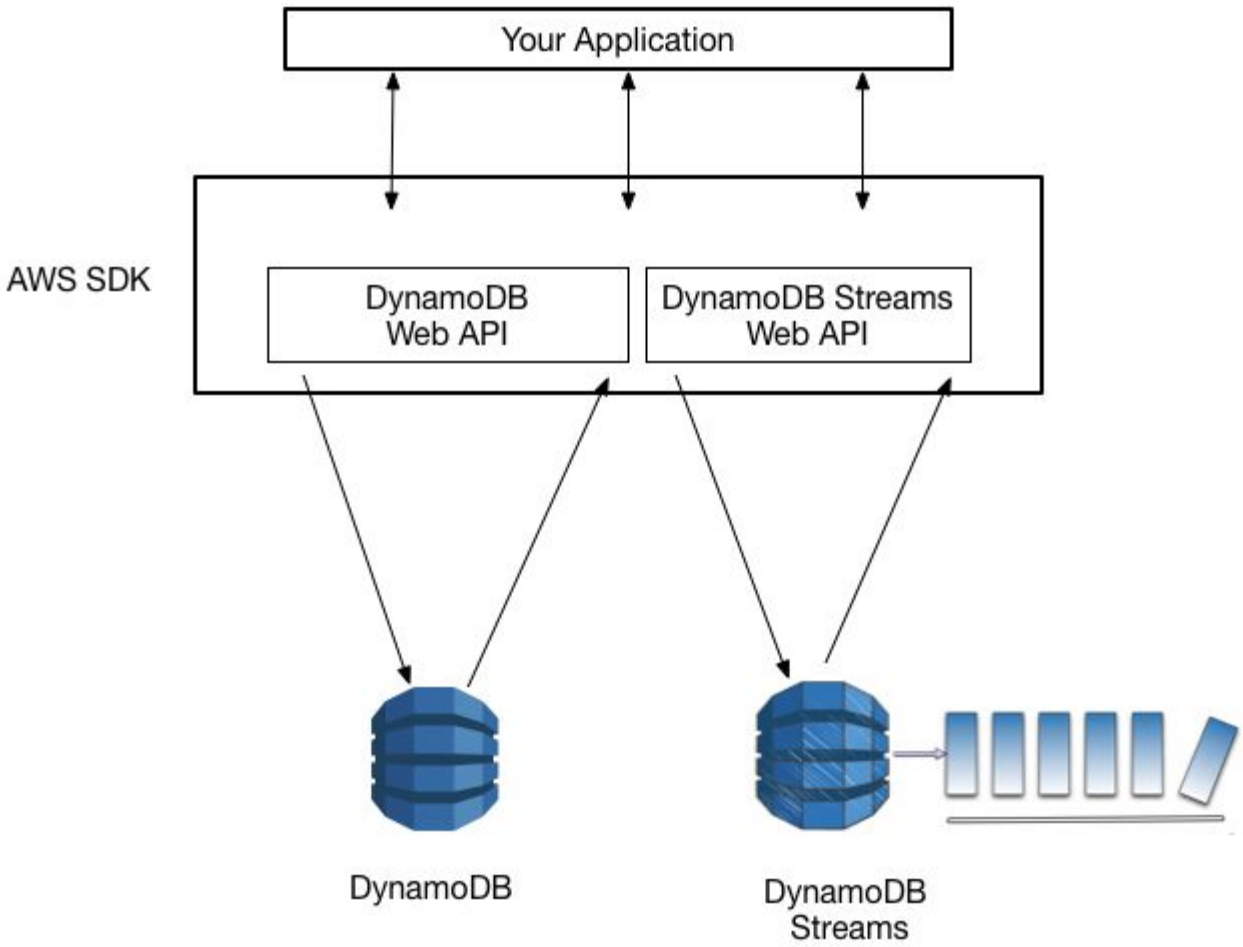
How to handle such use cases ?

- A global multi-player game has a multi-master topology, storing data in multiple AWS regions. Each master stays in sync by consuming and replaying the changes that occur in the remote regions.
- An application automatically sends notifications to the mobile devices of all friends in a group as soon as one friend uploads a new picture.
- A new customer adds data to a DynamoDB table. This event invokes another application that sends a welcome email to the new customer.

Understanding Streams

- Captures time ordered sequence of item level modification in any dynamo table
- Stores information in a log for upto 24 hours
- Once can see before and after data in near real time
- “**DynamoDB Stream -** is ordered flow of information about changes to Items in a dynamodb table”
- “**Stream Record” -** info about a data modification to a single item
- Streams can be enabled/disabled at any time
- Once can configure what gets recorded
  - KEYS Only
  - NEW image
  - OLD image
  - New and Old Image

DynamoDB Endpoints



Capturing Table Activity

- If you need **ability to capture changes to items** stored in a DynamoDB table, **at the point in time when such changes occur** then use **streams**

TTL

DynamoDB Accelerator (DAX)

End of Module

Q & A