PHP Web Services

Apache Cassandra

Presented by: Sara Kassani

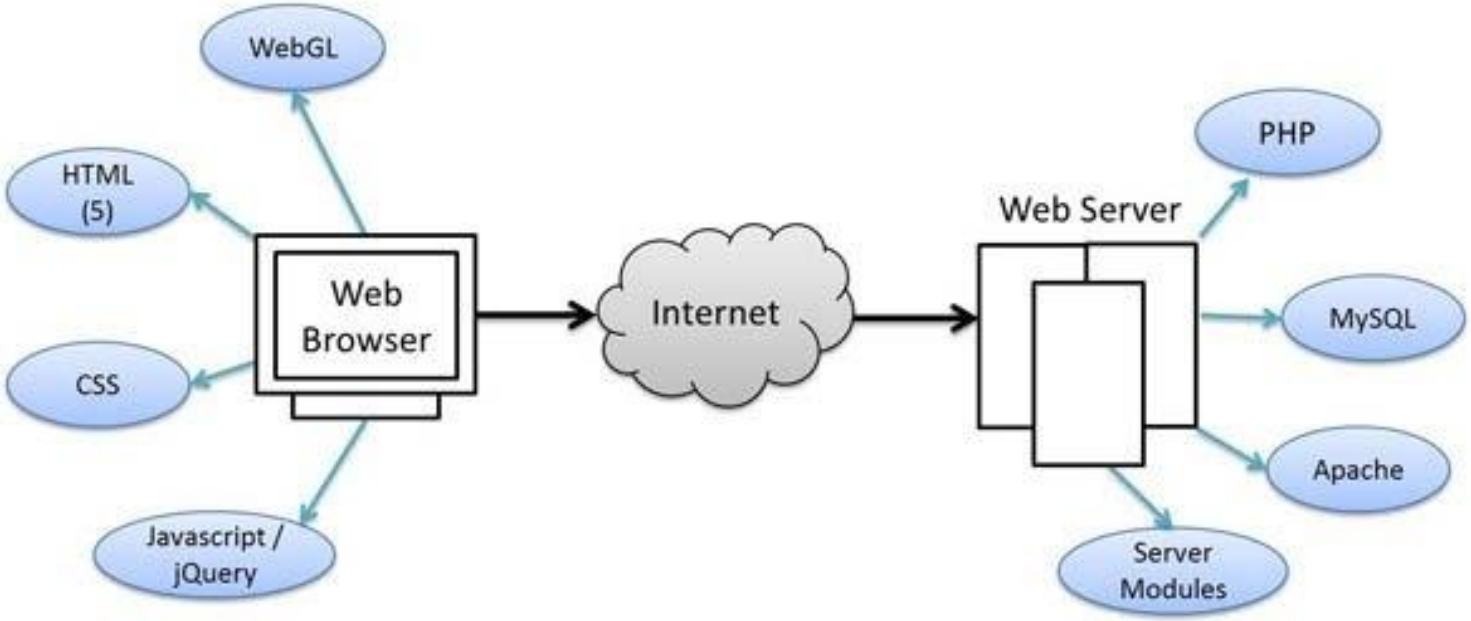
REST

- REST is about resources and how to represent resources in different ways.
- REST is an architecture all about the Client-Server communication
- REST is about how to manipulate resources.
- REST offers a simple, interoperable and flexible way of writing web services that can be very different from other techniques.

REST

- **Re**presentational **S**tate **T**ransfer
- Its all about client and server operations
- Idea: a network of web pages where the client progresses through an application by selecting links
- When client traverses link, accesses new resource (i.e., transfers state)
- Uses existing standards, e.g., HTTP

The Web


WebGL
HTML (5)
CSS
Javascript / jQuery
Web Browser
Internet
Web Server
PHP
MySQL
Apache
Server Modules

REST

- Client **requests** a specific **resource** from the server.
- The server **responds** to that request by delivering the requested resource.
- Server does not have any information about any client.
- So, there is no difference between the two requests of the same client.

The Fundamentals

- **Everything is a resource**
- **Every resource is identified with a unique identifier, URI**
- **REST uses simple and uniform interfaces like HTTP methods**
- **Stateless**

REST is NOT!

- A protocol.
- A standard.
- A replacement for SOAP.
  - SOAP has its own ways.

REST Characteristics

- Resources: Application state and functionality are abstracted into resources.
  - *URI*: Every resource is **uniquely addressable** using URIs.
  - *Uniform Interface*: All resources share a uniform interface for the transfer of state between client and resource
  - Methods: Use only HTTP methods such as **GET**, **PUT**, **POST**, **DELETE**, **HEAD**
  - Representation

URI Examples

- http://localhost:9999/restapi/books
  - GET – get all books
  - POST – add a new book
- http://localhost:9999/restapi/books/{id}
  - GET – get the book whose id is provided
  - POST – update the book whose id is provided
  - DELETE – delete the book whose id is provided

RESTful Web Services

- RESTful web services are web services which are REST based.
- **Stateless** & **cacheable**.
- Uses **URI** & **HTTP** methods.
- Frequently used with **SOA** projects.
- Quiet light, extensible and simple services.
- The reason behind the **popularity of REST** is that the applications we use are **browser-based** nowadays and top it all, REST is built on **HTTP**.
- Main idea: Providing the communication between **client** and **server** over **HTTP** protocol rather than other complex architectures like SOAP and RPC etc.

Building PHP RESTful Web Services

Building a simple API like Twitter

There will be a single route
That lets users retrieve a list
of messages

USERS

Send *messages*
Retrieve *messages*

MESSAGES

Sent by *users*
Retrieved by *users*

Each message will include a body,
sender, and datetime

Configuring and Installing Required Packages

- **Installing WAMP**
- **Installing Composer**
- **Installing Silex**

Installing Composer

getcomposer.org

Silex

- Open a command prompt
- Run the command below in the project directory, on the command line
  - composer require silex/silex "~2.0"
- Later, composer must have created some folders and some files like the figure below in the same directory for you
- Go and check

Silex

- **Create a user (/user) - This will use POST to create a new user in the DB.**
- **Update a user (/user/{id}) - This will use PUT to update the user**
- **View a user (/user/{id}) - This will use GET to view the users information**
- **Delete a user (/user/{id}) - This uses the DELETE method.**

Route

A route is simply the URL, and HTTP verb pattern that we want o match, and process

- **Get/messages**
- **POST/message**
- **POST/user**
- **DELETE/message/$messageId**

These routes below look alike, but they aren't the same

GET/messages

GET/messages/$messageId

Creating Our First Route

C:\wamp64\www\Tweetify\index.php - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

OPEN FILES

index.php — hello_silex

index.php — Tweetify

index.php — Tweetify

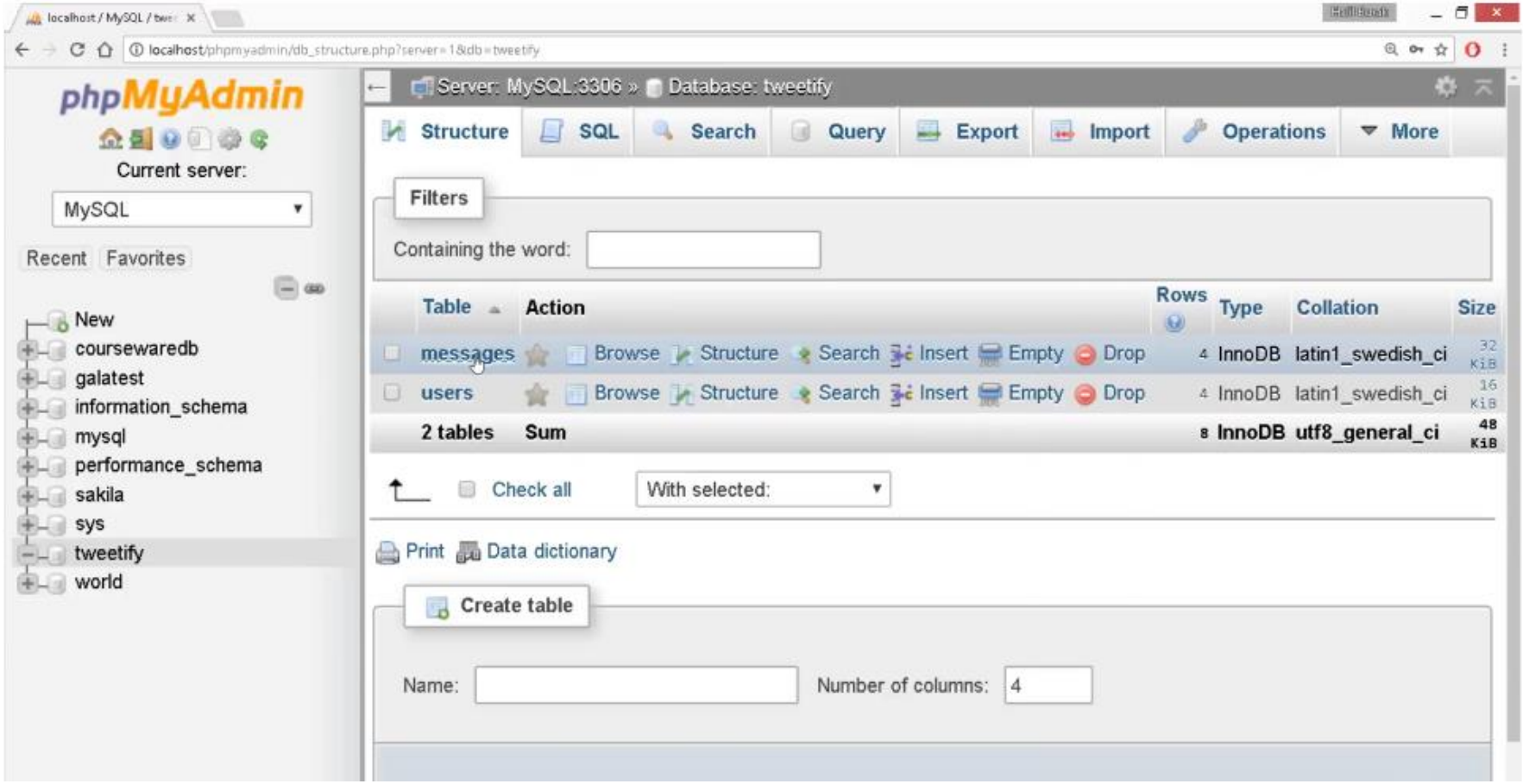
```php
<?php

require 'vendor/autoload.php';

$app = new Silex\Application();

$app->get('/messages', function() use($app) {
    return "This is a list of messages.";
});

$app->run();
```

Creating Database, Tables, and Middleware Component

So far

- **We have a route**
  - **Get messages**
- **But we aren't connected a database yet!**

Creating Database, Tables, and Middleware Component



Creating Database, Tables, and Middleware Component

localhost/phpmyadmin/tbl_structure.php?db=tweetify&table=users

phpMyAdmin

Current server:

MySQL

Recent Favorites

- New
- coursewaredb
- galatest
- information_schema
- mysql
- performance_schema
- sakila
- sys
- tweetify
- world

Server: MySQL:3306 » Database: tweetify » Table: users

Browse | Structure | SQL | Search | Insert | Export | Import | More

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(10)			No	None		AUTO_INCREMENT	Change
2	username	varchar(32)	latin1_swedish_ci		No	None			Change
3	password	varchar(64)	latin1_swedish_ci		No	None			Change
4	email	varchar(50)	latin1_swedish_ci		No	None			Change
5	profile_icon	varchar(255)	latin1_swedish_ci		No	None			Change
6	apikey	varchar(255)	latin1_swedish_ci		No	None			Change
7	created_at	datetime			No	None			Change
8	updated_at	datetime			Yes	NULL			Change

Check all *With selected:* Browse Change Drop Primary Unique Index

Print Propose table structure Move columns Improve table structure

Add 1 column(s) after updated_at Go

Creating Database, Tables, and Middleware Component

C:\wamp64\www\Tweetify\database\create.sql (Tweetify) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

OPEN FILES

FOLDERS
- Tweetify
  - database
    - create.sql
  - src
  - vendor
  - .htaccess
  - composer.json
  - composer.lock
  - index.php

create.sql

```sql
--
-- Table structure for table `messages`
--

CREATE TABLE `messages` (
  `id` int(10) NOT NULL,
  `user_id` int(10) NOT NULL,
  `body` varchar(255) NOT NULL,
  `image_url` varchar(255) NOT NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime NULL
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=

--
-- Dumping data for table `messages`
--

INSERT INTO `messages` (`id`, `user_id`, `body`,
(1, 1, 'Have you ever thought about life in the s
```

Establishing the Connection

C:\wamp64\www\Tweetify\bootstrap - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

OPEN FILES
- index.php — hello_slim
- index.php — Tweetify
- bootstrap

```php
<?php

include 'vendor/autoload.php';

use Illuminate\Database\Capsule\Manager as Capsule;

$capsule = new Capsule();
$capsule->addConnection([
    "driver" => "mysql",
    "host" => "localhost",
    "database" => "tweetify",
    "username" => "tweetify_api",
    "password" => "tweetify_api",
    "charset" => "utf8",
    "collation" => "utf8_general_ci",
    "prefix" => ""
]);

$capsule->bootEloquent();
```

CRUD Operations Mapped to HTTP Methods in RESTful Web Services

OPERATION	HTTP METHOD
Create	POST
Read	GET
Update	PUT or POST
Delete	DELETE

HTTP Codes

Code Range	Message	Description
1xx	Information	
2xx	Successful	
3xx	Redirection	
4xx	Client error	
5xx	Server error	

Creating Read-Write API

The Read-Only Route

GET/messages: This simply returns a list of messages.

Four methods in this API

GET: Retrieves data from the API's resources. This method never changes anything on the server.

POST: Adds new resources into the API. Might also update API's data

PUT: Updates API's data. This methods will always change data on the server.

DELETE: Deletes data from the API

Post a new Tweet

```
$app->post('/message', function(Request $request) use($app) {

    $_message = $request->get('message');

    $message = new Message();
    $message->body = $_message;
    $message->user_id = 1;

    $message->save()


    return new Response('Message created.', 200);
});
```

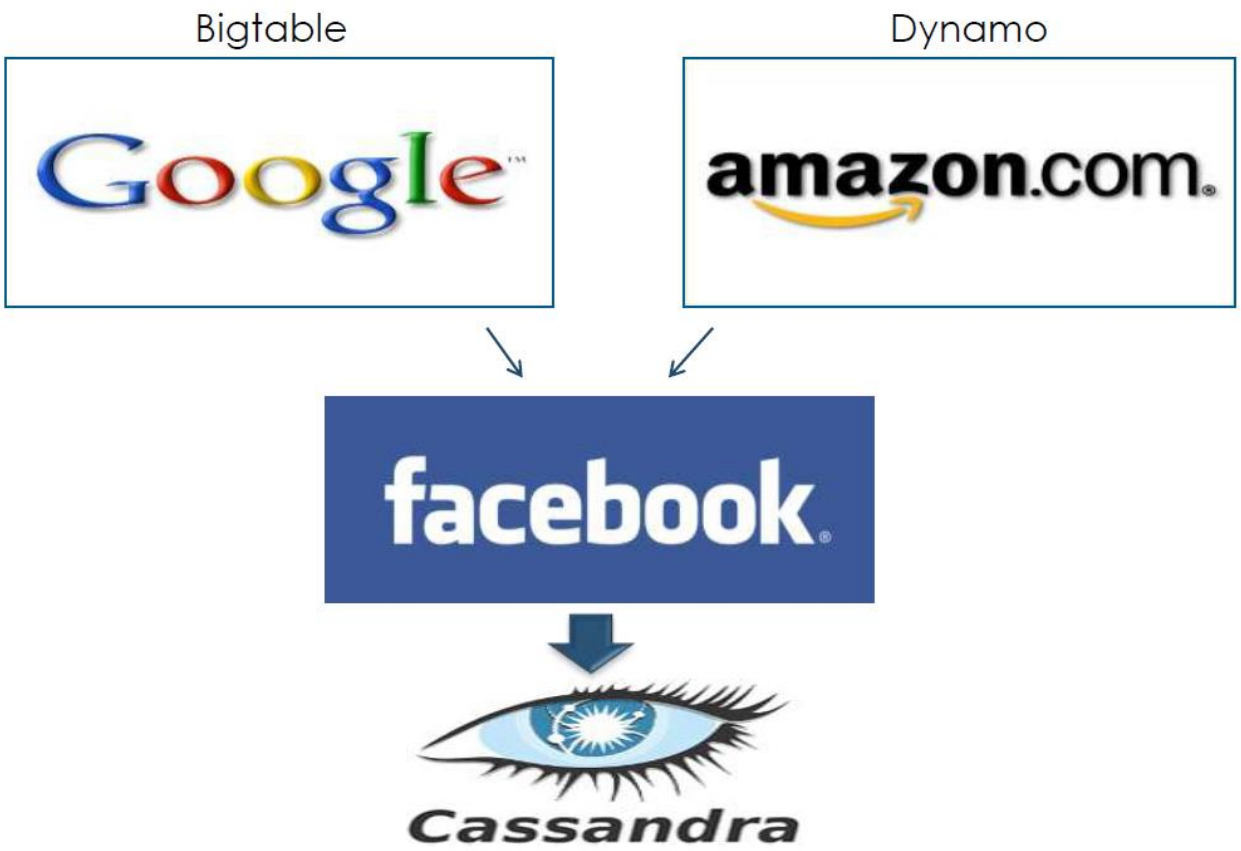
An Overview of Apache Cassandra

Presented by: Sara Kassani

Definition of Cassandra

- **Apache Cassandra™ is a free**
- **Distributed**
- **High performance**
- **Extremely scalable**
- **Fault tolerant (i.e. no single point of failure)**
- **Cassandra can serve as both real-time datastore for online/transactional applications, and as a read-intensive database for business intelligence**

The History of Cassandra

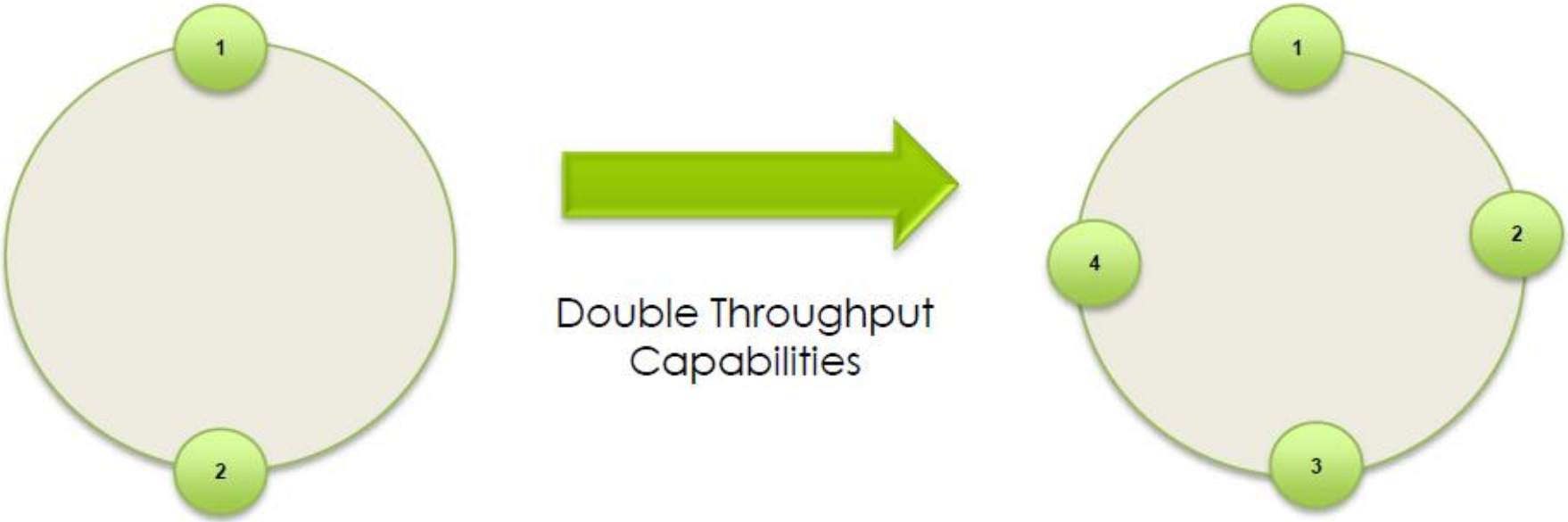


Architecture Overview

- **Cassandra was designed with the understanding that failures can and do occur**
- **Peer-to-peer, distributed system**
- **All nodes the same**
- **Read/Write-anywhere in any node**
- **Each node communicates with each other through the Gossip protocol, which exchanges information across the cluster every second**
- **A commit log is used on each node to capture write activity.**
- **Data also written to an in-memory structure (memtable) and then to disk once the memory structure is full (an SStable)**

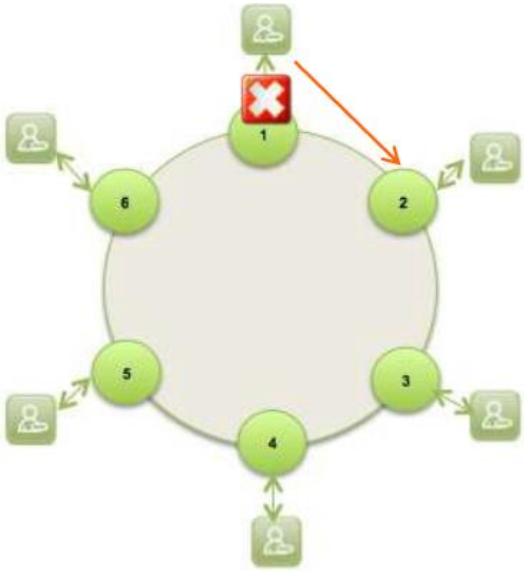
Big Data Scalability

- **Capable of comfortably scaling to petabytes**
- **New nodes = Linear performance increases**
- **Add new nodes online**



No Single Point of Failure

- **All nodes the same**
- **Customized replication affords tunable data redundancy**
- **Read/write from any node**
- **Can replicate data among differe**



RDBMS Architecture Overview

Database

Table1

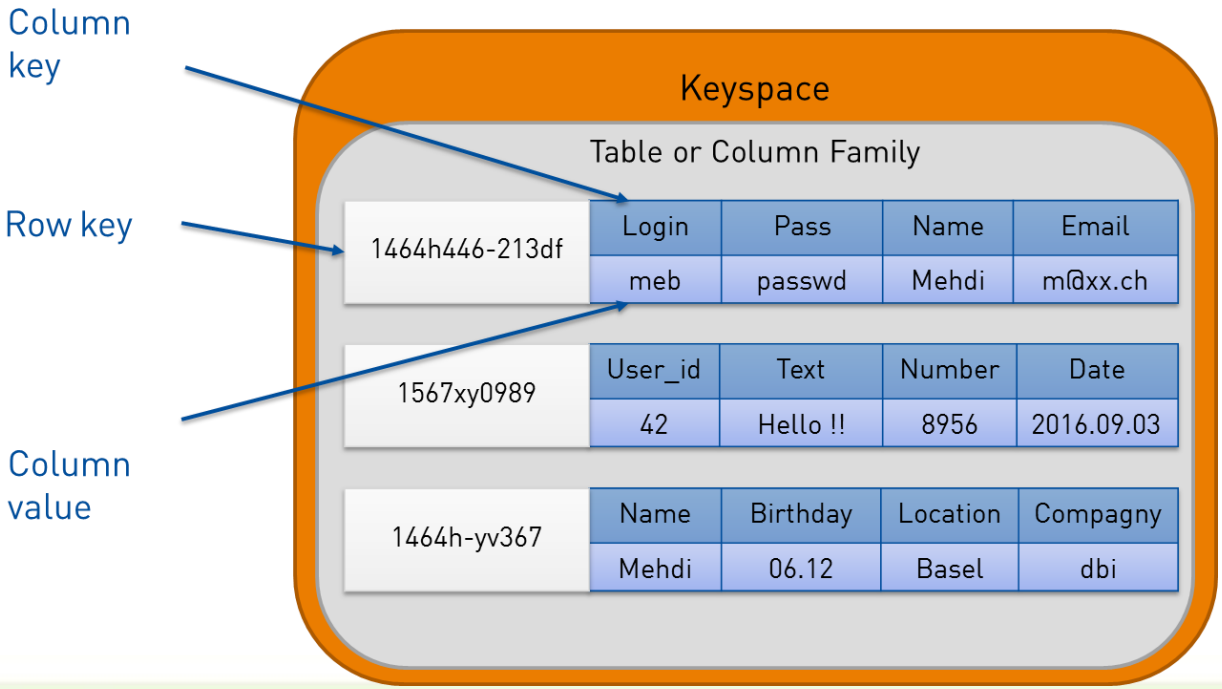
	Column1	Column2
Row1	value	value
Row2	*null*	value
...		

Table2

	Column1	Column2	Column3
Row1	value	value	value
Row2	*null*	value	*null*
...			

Cassandra Architecture Overview

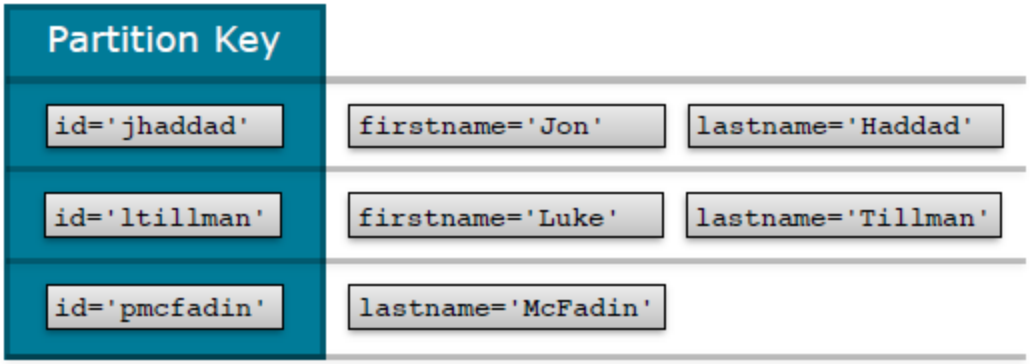
- **A keyspace is similar to a database in the RDBMS world**
- **A column family is similar to an RDBMS table but is more flexible/dynamic**
- **A row in a column family is indexed by its key. Other columns may be indexed as well**



Data Distribution

- **Partition Key determines node placement**

```
CREATE TABLE users (
    id text,
    firstname text,
    lastname text,
    PRIMARY KEY (id)
);
```



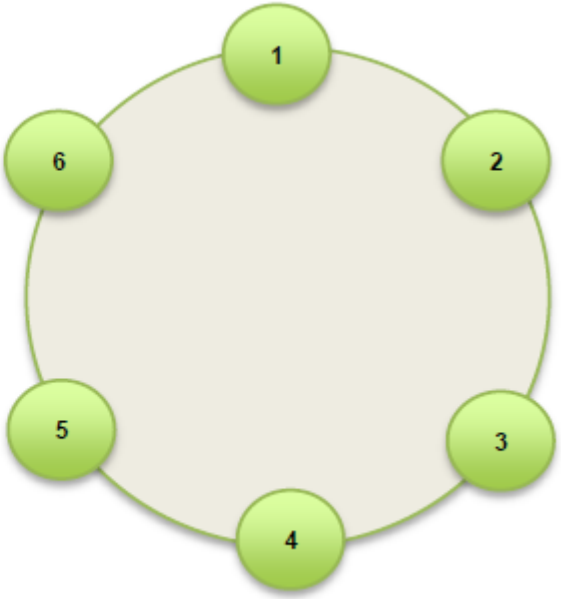
Replication Strategy

- **Simple Strategy – Use this for a single data center. It places the first replica on a node determined by the partitioner.**
- **NetworkTopologyStrategy - If you plan to have your cluster span across multiple data centers. Specifies how many replicas you want in each data center.**

CQL Language

- **Very similar to RDBMS SQL syntax**
- **Create objects via DDL (e.g. CREATE…)**
- **Core DML commands supported: INSERT, UPDATE, DELETE**

```
SELECT  *
FROM    USERS
WHERE   STATE = 'TX';
```



Defining Keyspaces in CQL

- **CREATE KEYSPACE users**
- **WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };**

- **CREATE KEYSPACE users**
- **WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3, 'dc2' : 2};**

Inserts and Updates

- **Use INSERT or UPDATE to add and modify data**

```
INSERT INTO comments_by_video (
    videoid, commentid, userid, comment)
VALUES (
    '0fe6a...', '82be1...', 'ac346...', 'Awesome!');
```

```
UPDATE comments_by_video
SET userid = 'ac346...', comment = 'Awesome!'
WHERE videoid = '0fe6a...' AND commentid =
'82be1...';
```

Datacenter

- **Grouping of nodes of data.**
- **Each data center can have separate replication settings.**
- **May be in different geographical locations, but not always.**

Cluster

- **Grouping of datacenters and nodes that communicate with each other and replicate data.**
- **Clusters are not aware of other clusters.**

Updates and Deletes

- **SSTable files are immutable and cannot be changed.**
- **Updates are written as new data.**
- **Deletes write a tombstone, which mark a row or column(s) as deleted.**
- **Updates and deletes are just as fast as inserts.**

SSTable

id:1, first: John,
last: Smith

timestamp: ...405

SSTable

id:1, first: John,
last: Williams

timestamp: ...621

SSTable

id:1, deleted

timestamp: ...999