

# Vyhľadávanie podobných obrázkov

## Apache Cassandra

eduard kuric

---



**PeWe@FIIT**  
personalized web group

# Agenda

1. Motivácia - reálny projekt
  - vyhľadávanie obrázkov
  - automatické anotovanie fotografií
1. Apache Cassandra
  - NoSQL úložisko: kľúč-hodnota (key-value storage)

# Motivácia - vyhľadávanie

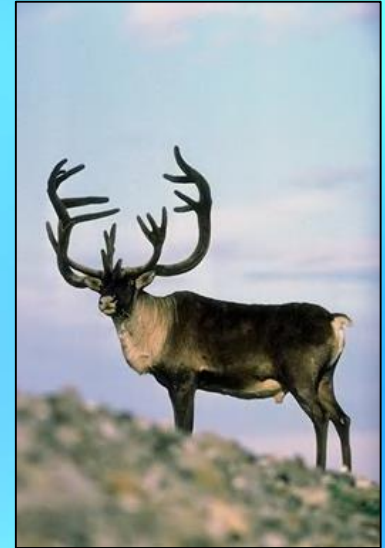
- Každý z nás už pravdepodobne niekedy vyslovil podobnú vetu:

„Rád by som ti tú fotografiu ukázal,  
ale nedokážem ju v rýchlosti nájsť.“

- Prečo?
  - používatelia vyhľadávajú spravidla pomocou kľúčových slov, ALE: obrázky sú vytvorené z pixelov – neobsahujú jednotky akými sú slová v porovnaní s textovými dokumentmi

# Prečo automatické anotovanie obrázkov?

- Manuálne anotovanie fotografií:
  - časovo náročné
  - subjektívne
  - pre niekoho je na fotografii jeleň, pre iného caribou
- Preto sa už viac ako dekádu venuje výskum automatickému anotovaniu obrázkov (fotografií)



# 1. scenár automatického anotovania

- Obrázok je vložený v dokumente (obklopený textom)
  - získavanie anotácií: ALT, bezprostredné okolie, názov súboru,...
- Google obrázky
  - – dopyt „car“ - prvé dva výsledky (súčasnosť):



*súbor:* Solar\_Wing\_front\_Japanese\_electric\_powered\_**car**.jpg

*ALT:* The Solar Wing, Japanese electric racing **car**

*okolie:* WHAT IS A SOLAR **CAR**, A solar **car** is...

# 1. scenár automatického anotovania /2

- Google obrázky, rok 2004
  - získavanie anotácií: ~~ALT, bezprostredné okolie~~, názov súboru
- dopyt: „car“ = mapa Chicaga
  - <http://maps.uchicago.edu/directions/graphics/car.gif>



Ahn, L., Dabbish, L.: Labeling images with a computer game. In Proc of the SIGCHI conference on Human factors in computing systems (CHI '04). ACM, New York, 2004, pp. 319-326.

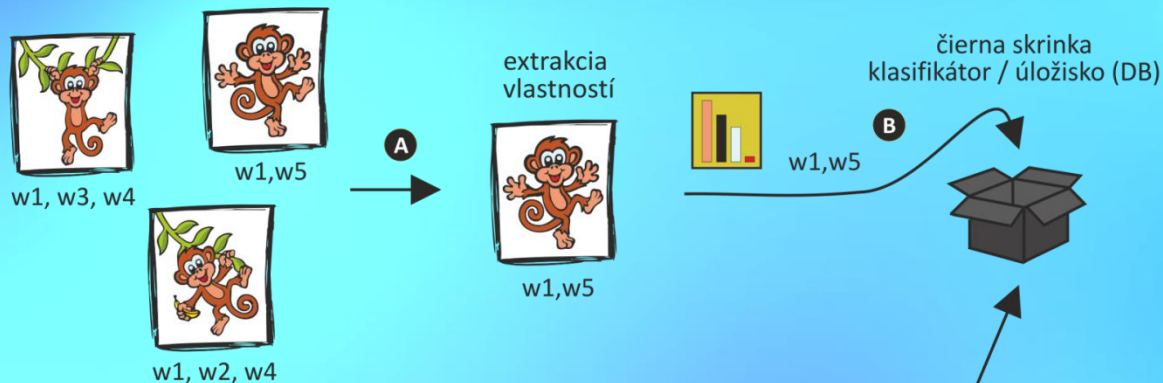
## 2. scenár automatického anotovania

- Obrázok bez akéhokoľvek kontextu (101.jpg)  
(foto album, archív na pevnom disku, ...)
  - vstup: obrázok (fotografia) bez tagov
  - výstup: fotografia s priradenými tagmi,  
ktoré ju opisujú vizuálny obsah (konkrétne/všeobecne)
- **POŽIADAVKA: robustný trénovací dataset**



# Všeobecný model

spracovanie  
„trénovacieho“ datasetu



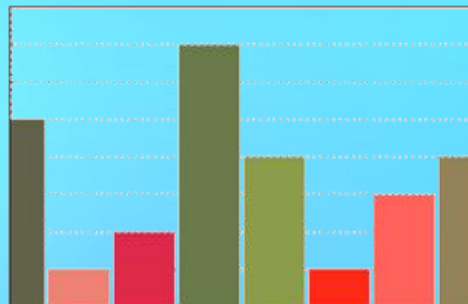
spracovanie  
cieľovej fotografie





# Extrakcia vlastností

Globálne vlastnosti



Lokálne vlastnosti (feature points)



# Kombinácia vlastností

- JCD (Joint Composite Descriptor)
- SIFT (Scale Invariant Feature Transform),  
alter. SURF (Speeded Up Robust Features)
- Prečo skombinovať?
  - problém: lokálne vlastnosti  
homogénne oblasti napr. voda, obloha,  
piesok

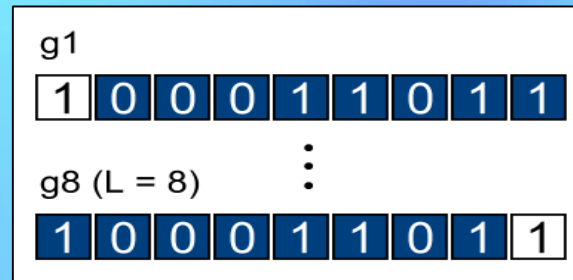


# Lokálne vlastnosti – veľký problém

- Ak chceme robustný (trénovací) korpus, ktorý bude obsahovať potenciálne milióny fotografií, vzniká nám pri použití lokálnych vlastností veľký problém:
- Z jednej fotografie môže byť extrahovaných stovky až tisíce vlastností
  - Ako ich navzájom porovnávať v reálnom čase?
  - Ako ich uložiť tak, aby sme k nim dokázali pristupovať v reálnom čase?

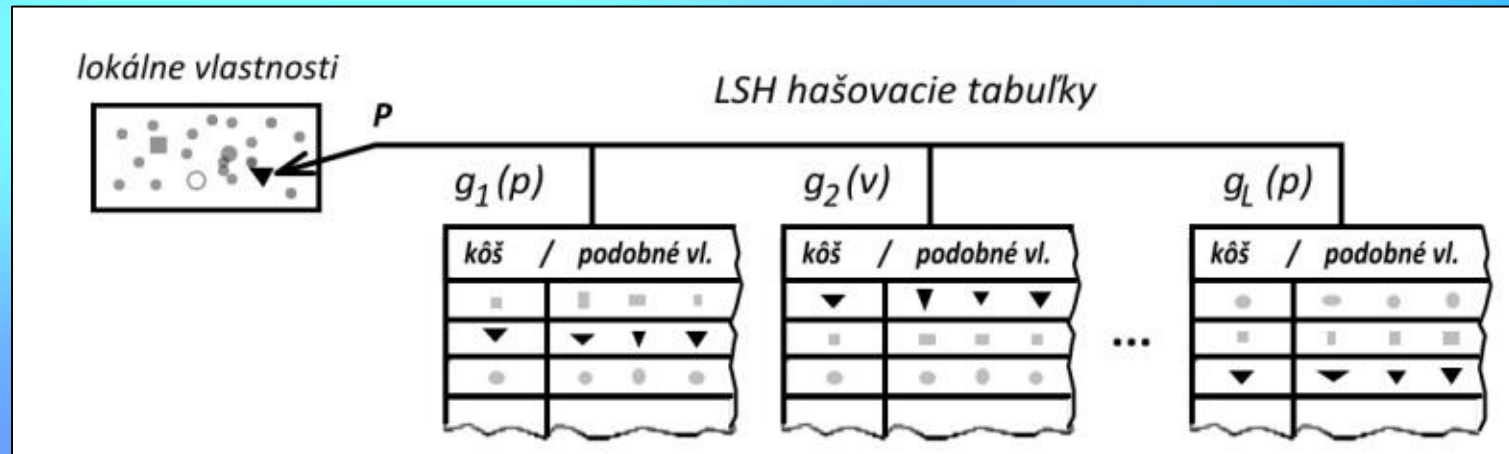
# Ako ich navzájom porovnávať v reál. čase?

- **LSH** - Lokálne senzitivné hašovanie  
(angl. Locality sensitive hashing)
- **Objekty**, ktoré sú si **podobné** sú vložené (zahašované) do **rovnakého koša** (angl. bucket) s vysokou pravdepodobnosťou
  - Rodina hašovacích funkcií ( $g_1, \dots, g_L$ )
  - Každá funkcia  $g$  vznikne spojením (konkatenáciou)  $k$  hašovacích funkcií:



# Lokálne Senzitívne Hašovanie

- Každéj hašovacej funkcii  $g$  prislúcha jedna hašovacia tabuľka:
  - riadok v tabuľke: kľúč koša / podobné vlastnosti



# Lokálne Senzitívne Hašovanie (LSH) / 3

BucketID	ImageID_x_y	ImageID_x_y	...
1	1_135_11	5_41_31	...
2	2_56_201	5_185_39	...
...	...	...	...

ImageID	Keypoint Location (x_y)			...		
	Descriptor	Orientation	Size	...	...	...
1	135_11			...		
	$[A_1, \dots, A_{128}]$	B	C	...	...	...
2	56_201			...		
	$[X_1, \dots, X_{128}]$	Y	Z	...	...	...
...	...			...		
	...	...	...	...	...	...

# Ako efektívne uložiť hašovacie tabuľky?

- Požiadavky:
  - Index v pamäti, zvyšok pevný disk
    - LSH memory-based
  - Škálovateľnosť
    - jedna hašovacia tabuľka / počítač,
    - jedna hašovacia tabuľka / viacero počítačov
- Riešenia:
  - navrhnuť a implementovať „disk-based“ LSH
  - DBMS Apache CASSANDRA



# Apache Cassandra

- OpenSource - špeciálne navrhnutý pre spracovanie veľkého objemu údajov, distribuovaný decentralizovaný hybrid medzi stĺpcovo a riadkovo orientovaným DBMS

**Amazon Dynamo**

(architektúra)



- DHT (distribuovaná hašovacia tabuľka)

**Google BigTable**

(datový model)



- Columns families, Columns

# Apache Cassandra – hybridná orientácia

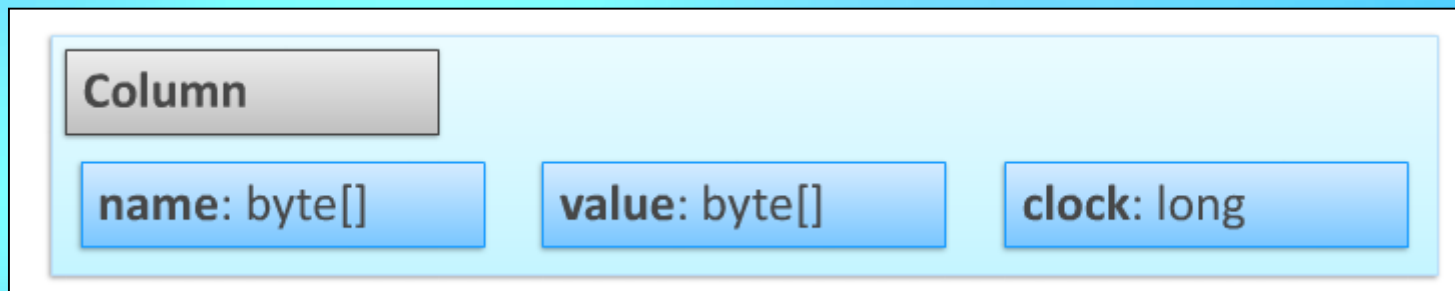
- Stĺpcova (Columns) orientácia
  - počet stĺpcov nie je fixný
  - stĺpce môžu byť usporiadané
  - stĺpce môžu byť dopytované na určitý rozsah
- Riadková (Row) orientácia
  - každý riadok je jednoznačne identifikovateľný kľúčom
  - riadky zoskupujú stĺpce a super stĺpce

# Apache Cassandra – keyspace

- V hantírke relačných DBMS = databáza
- Vlastnosti
  - replikačný faktor
  - stratégia replikácie
  - viacero rodín stĺpcov (Column Families) = tabuľky
- Pre jednu aplikáciu je možné vytvoriť viacero keyspaces (napr. pri potrebe rôznych replikačných stratégií)

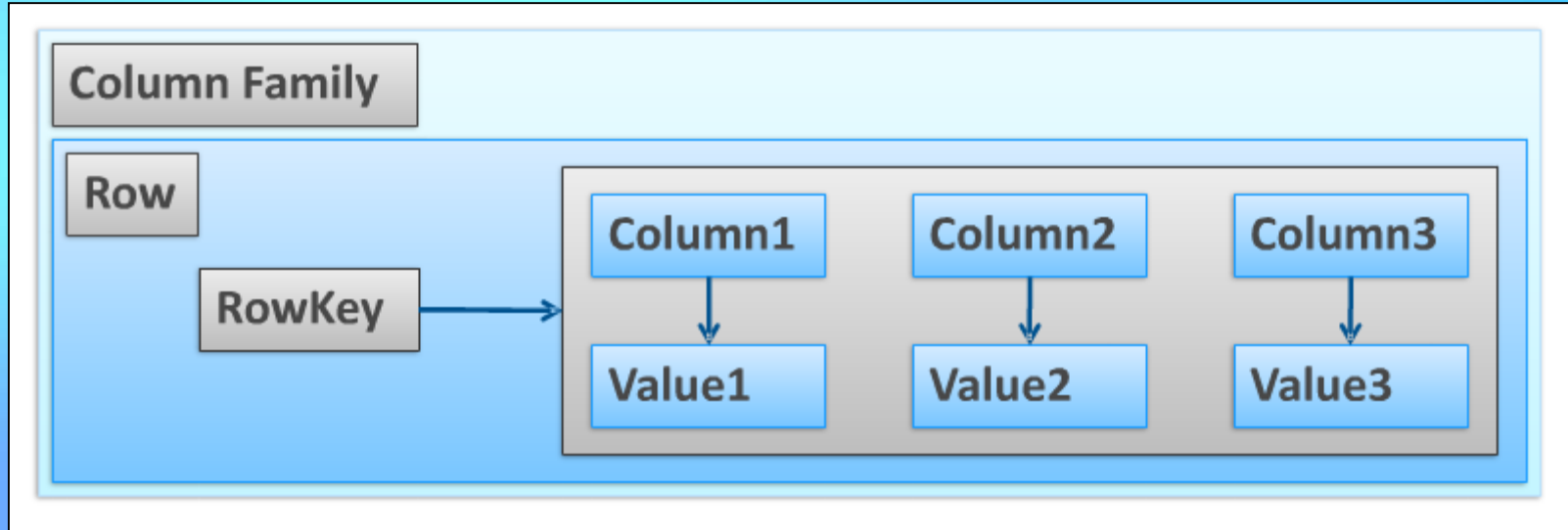
# Apache Cassandra – column

- Základná jednotka údajovej štruktúry



# Apache Cassandra – column family

- V hantírke relačných DBMS = tabuľka
- Kontajner pre kolekciu riadkov

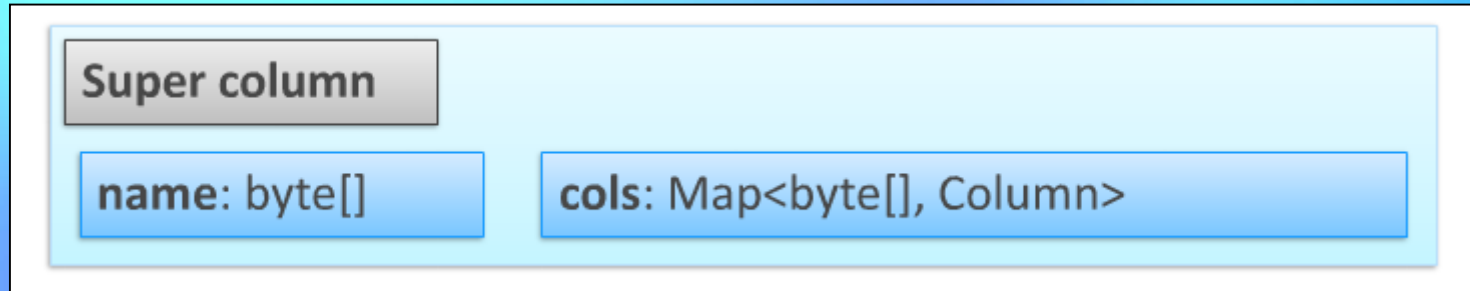


# Apache Cassandra – column family /2

- 4-dimenzionálne mapovanie  
[Keyspace][ColumnFamily][Key][Column]
- Počet stĺpcov nie je striktne definovaný
- Column môže byť SuperColumn
- CF má komparátor atribút – ako sú usporiadané výsledky pri dopyte
- Každá CF je uložená v separátnom súbore  
(užitočné pri ukladaní príbuzných stĺpcov v rovnakej CF)

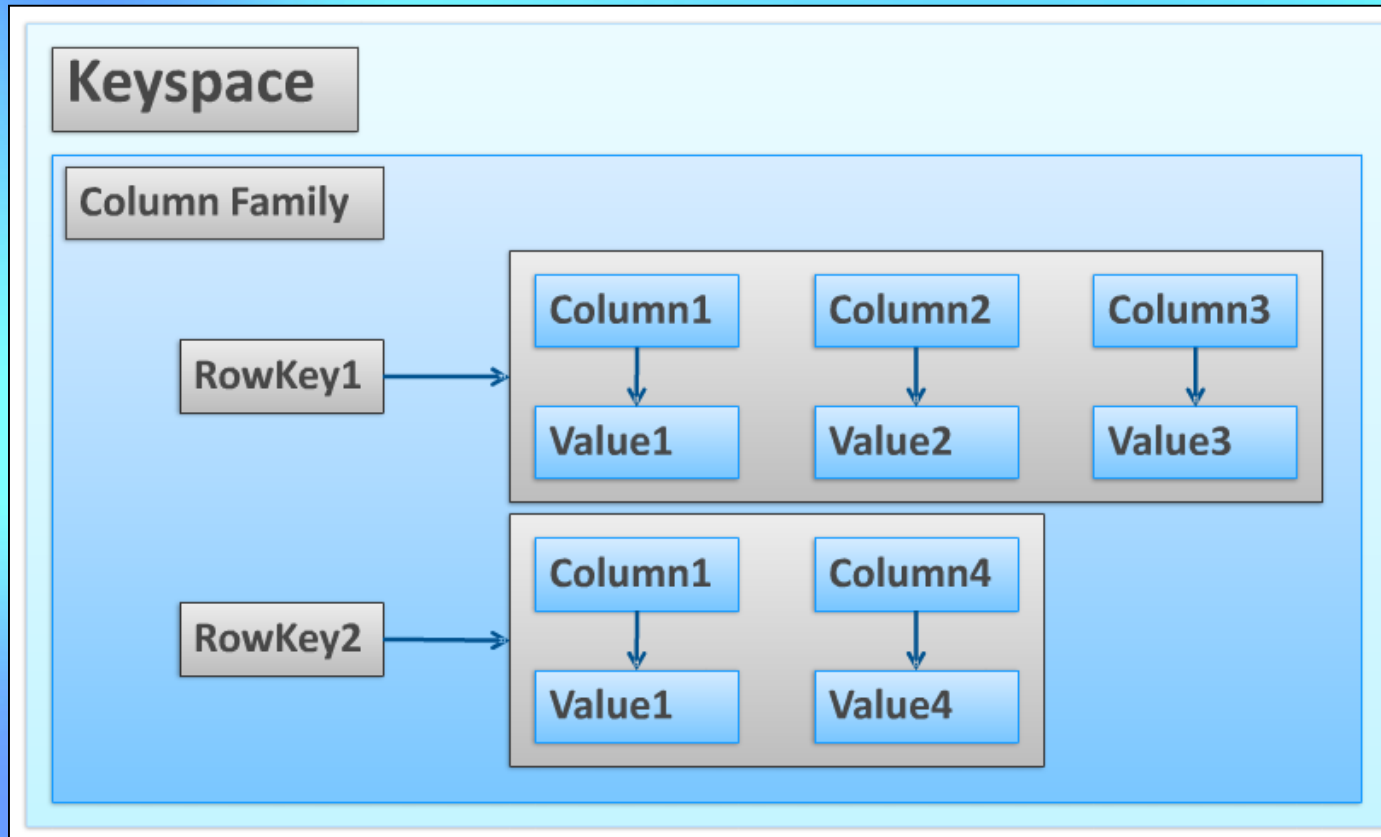
# Apache Cassandra – super columns

- 5-dimenzionálne mapovanie
  - [Keyspace][ColumnFamily][Key][SuperColumn][SubColumn]
- maximálne jednoúrovňová hĺbka (SubColumn nemôže byť SuperColumn)





# Apache Cassandra – dátový model



# Apache Cassandra – wide/skinny rows

- Wide rows – veľký počet stĺpcov (Columns) a malý počet riadkov
  - zle funguje s RowCache
- Skinny rows – malý počet stĺpcov (Columns) a veľký počet riadkov
- Ak máte veľký počet riadkov a veľký počet stĺpcov, tak docielite veľké indexy
  - ~40GB dát = 10GB index

# Apache Cassandra – typy komparátorov

- AsciiType
- BytesType
- IntegerType
- LongType
- TimeUUIDType
- UTF8Type

# Apache Cassandra – replikácia

- Ktorý uzol je použitý na uloženie riadku je určené **mapovaním** jeho **klúča** na hodnotu **tokenu**, ktorú určí **partitioner (DHT)**
- Každý server (uzol) je schopný uložiť tokeny v určitom rozsahu

Key	Map to	Server o	init_token	Range responsible
		0	0	$2^{**127/4} * 3 .. 0$
"somekey"	45678845	1	$2^{**127/4}$	$0 .. 2^{**127/4}$
		2	$2^{**127/4} * 2$	$2^{**127/4} .. 2^{**127/4} * 2$
		3	$2^{**127/4} * 3$	...

# Apache Cassandra – konzistencia údajov

- Replikačný faktor (počet kópií)
- Úroveň konzistencie (počet replík na prístup každej read/write operácie)

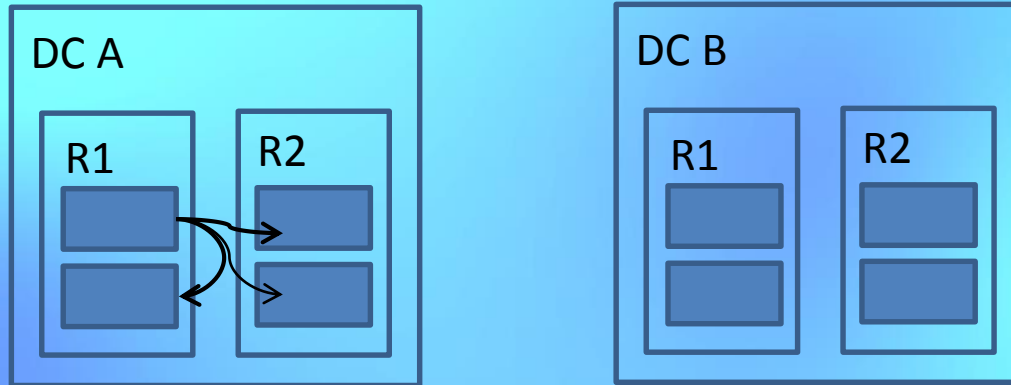
Consistency level	Read / Write
ONE	1 replica
QUORUM	$N/2 + 1$
ALL	N

# Apache Cassandra – konzistencia údajov

- READ operácia
  - pre každý READ sú dopytované všetky repliky
  - ak je v niektorej z replík nezhoda
    - pull data -> merge -> sync
- weak vs strong
  - weak: vykoná opravu PO vrátení výsledkov
  - strong: vykoná opravu PRED vrátením výsledkov

# Apache Cassandra – stratégie replikácie

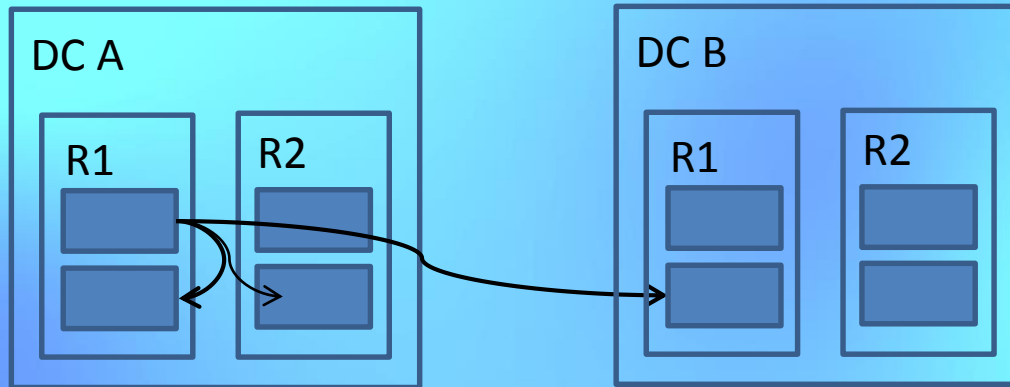
- Typ stratégie určuje, ktoré ďalšie uzly sú vybraté od uzla, ktorý je daný hodnotou tokenu
  - **RackUnaware** (default) – uzly, ktoré sú vedľa seba v kruhu (ignoruje topológiu)





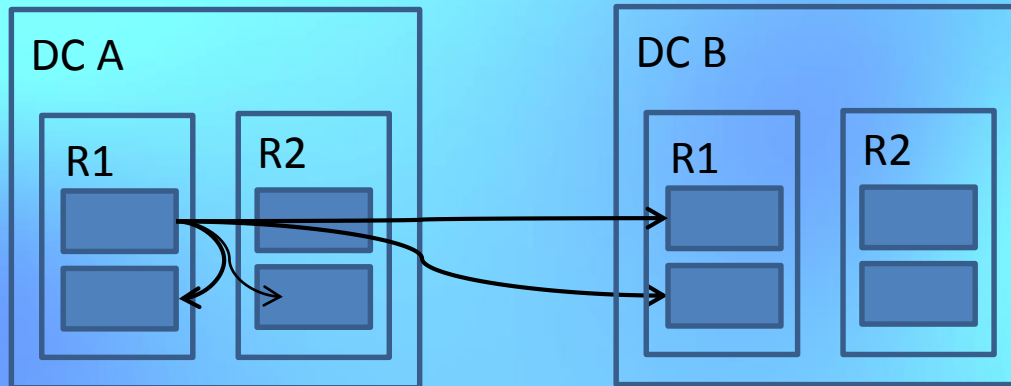
# Apache Cassandra – stratégie replikácie

- Typ stratégie určuje, ktoré ďalšie uzly sú vybraté od uzla, ktorý je daný hodnotou tokenu
  - **RackAware** – druhú repliku do iného DC, zvyšných  $N-2$  replík do uzlov v iných rackoch v rovnakom DC



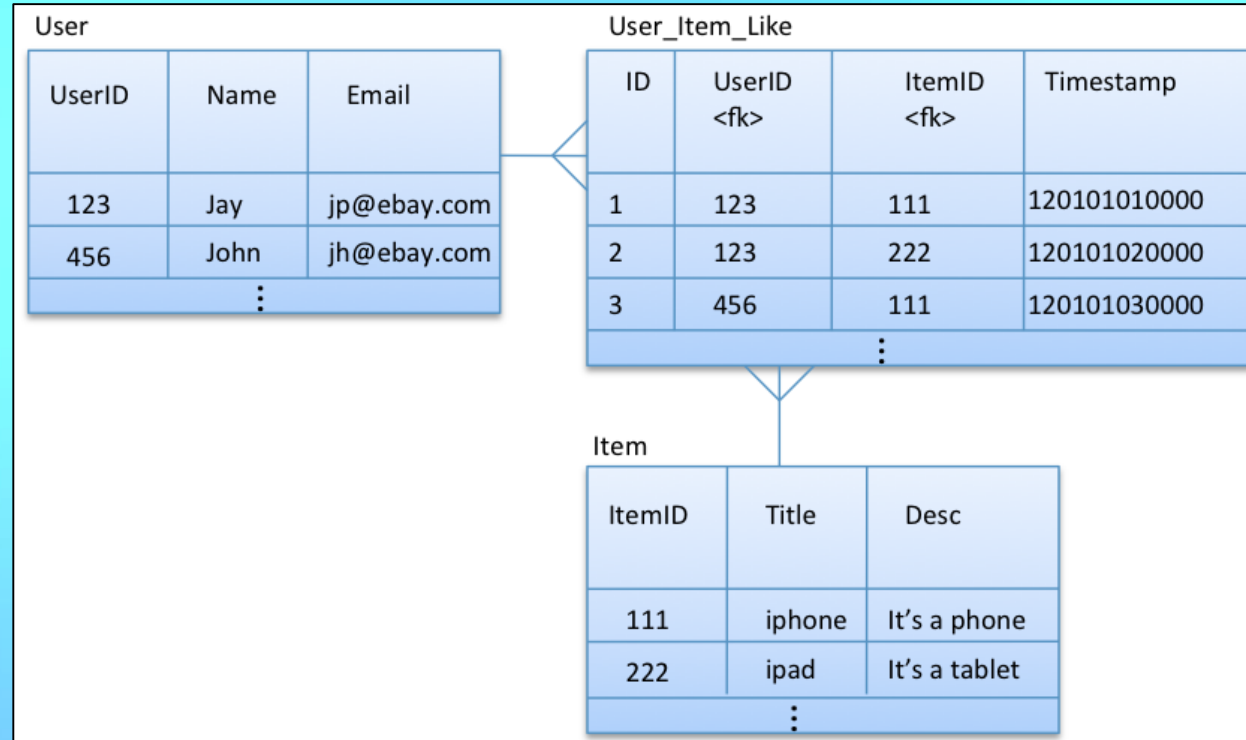
# Apache Cassandra – stratégie replikácie

- Typ stratégie určuje, ktoré ďalšie uzly sú vybraté od uzla, ktorý je daný hodnotou tokenu
  - **DatacenterShard** – M z N replík do iného DC, zvyšných N-M+1 replík do uzlov v iných rackoch v rovnakom DC



# Apache Cassandra – Príklad

1. Používateľa podľa ID
2. Položku podľa ID
3. Všetky položky, ktoré nejaký používateľ like-oval
4. Všetkých používateľov, ktorí like-ovali nejakú položku



# Apache Cassandra – Príklad riešenie


User			Item		
123	Name	Email	111	Title	Desc
	Jay	jp@ebay.com		iphone	It's a phone
⋮			⋮		

User_By_Item				
111	120101010000   123	120101030000   456	...	
	Jay	John		
⋮				

Item_By_User				
123	120101010000   111	120101020000   222	...	
	iphone	ipad		
⋮				

 <timeuuid|userid>

# Apache Cassandra – záver

- pri dopytovaní nie je možné výsledky joinovať (klient)
- TTL column typ = expiruje po určitom čase (napr. pre uloženie session token)
- MySQL vs Cassandra (50GB dát, priemer)
  - MySQL
    - Write ~300ms
    - Read ~350ms
  - Cassandra
    - Write ~0.12ms
    - Read ~15ms

# Apache Cassandra – záver / 2

- Apache Cassandra
  - <http://cassandra.apache.org/>
- Fluent Cassandra (.Net)
  - <https://github.com/managedfusion/fluentcassandra>
- Hector, Pelops (Java)
  - <http://github.com/rantav/hector>
  - <http://code.google.com/p/pelops/>
- Cassandra GUI
  - <http://code.google.com/a/apache-extras.org/p/cassandra-gui/>