



TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies



CASSANDRA

Lukáš Matějů

15.5.2024 | DPB



ČÁST I.: OPAKOVÁNÍ



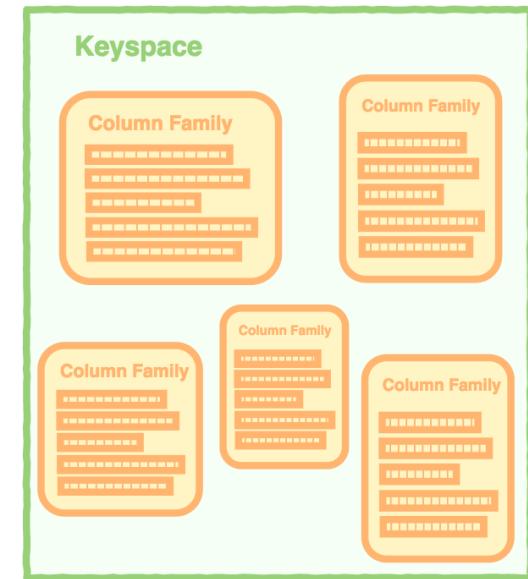
OPAKOVÁNÍ

- sloupcové databáze
 - distribuované NoSQL databáze
 - označovány také jako
 - sloupcově orientované databáze
 - sloupcově orientovaný model pro ukládání dat
 - data ukládána do záznamů schopných pojmut velké množství dynamických sloupců
 - klíče záznamů ani jména sloupců nejsou fixní
 - často považovány za dvoudimenzionální key-value úložiště
 - optimalizované pro rychlý přístup k velkým objemům dat
 - analytické aplikace, big data
 - výhody sloupcových databází
 - horizontální škálování a replikace
 - volné schéma
 - rychlé načítání dat, dotazování a agregace



OPAKOVÁNÍ

- sloupcově orientovaný model
 - základem je **keyspace**
 - volně odpovídá schématu v relačních databázích
 - dává představu o struktuře databáze
 - **keyspace** obsahuje **column families**
 - volně odpovídají relacím v relačních databázích
 - **column family** obsahuje seřazené **řádky** (rows)
 - jednoznačně identifikovány klíčem (row key)
 - každý řádek obsahuje **sloupce** (columns)
 - sloupce patří jen svému řádku
 - na rozdíl od relačních databází
 - počet sloupců se mezi řádky může lišit
 - stejně tak se mohou lišit názvy sloupců i jejich datové typy
 - každý sloupec obsahuje pár **jméno – hodnota** a **časovou značku**



<https://database.guide/what-is-a-column-store-database/>



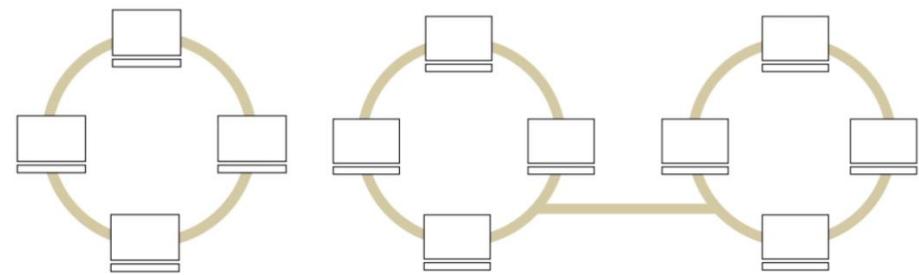
OPAKOVÁNÍ

- Cassandra
 - sloupcová NoSQL databáze od Apache
 - open source
 - napsaná v Javě
 - založená na Amazon Dynamo a Google Big Table
 - distribuovaná
 - horizontální (lineární) škálování
 - replikace
 - technologie pro big data
 - vysoká dostupnost
 - velmi často databáze na velkém množství uzlů přes více datových center
 - odolnost vůči chybám
 - každý uzel má stejnou funkci (vs. master – slave)

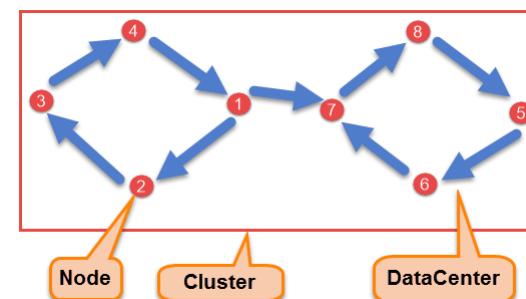


OPAKOVÁNÍ

- architektura Cassandra
 - základem je **uzel** (node)
 - ukládá data
 - každý uzel plní stejnou roli
 - neexistují slaves ani master
 - uzly jsou replikované
 - neexistuje jeden bod selhání
 - podpora virtuálních uzlů
 - příbuzné uzly tvoří **datová centra**
 - fyzické i virtuální
 - replikace na úrovni datových center
 - prstencová struktura
 - datová centra tvoří **cluster**
 - dostupnost dat i při výpadku centra



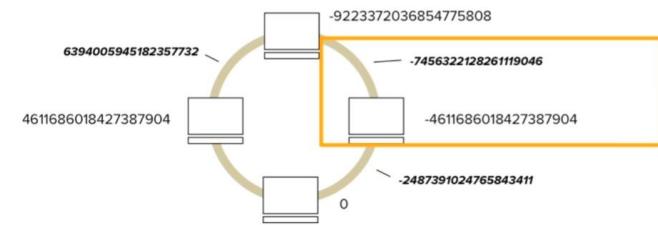
<https://www.udemy.com/course/apache-cassandra>



<https://www.guru99.com/cassandra-architecture.html>

OPAKOVÁNÍ

- architektura Cassandra
 - uzly o sobě vědí díky **snitch**
 - poskytuje uzlům informaci o topologii sítě
 - definuje skupiny uzelů do datových center a racků
 - replikační strategie využívá skupiny pro umístění replik
 - SimpleSnitch, GossipingPropertyFileSnitch
 - uzly spolu komunikují pomocí **gossip**
 - peer-to-peer komunikační protokol pro zjišťování a sdílení informací o ostatních uzlech
 - každou vteřinu každý uzel komunikuje až se třemi dalšími uzly
 - uzly si vyměňují informace o sobě a o všech ostatních uzlech, o kterých mají informace
 - o distribuci dat se stará **partitioner**
 - pomocí konzistentního hashování
 - využívá algoritmus přidělující každému řádku odpovídající partition (token a uzel)
 - v praxi využívá virtuální uzly
 - o replikaci se stará **replikační faktor** a **replikační strategie**



<https://www.udemy.com/course/apache-cassandra>

OPAKOVÁNÍ

- architektura Cassandra
 - v Cassandře je databáze definována jako **keyspace**
 - keyspace obsahuje **tabulky** (tables)
 - všechna data mají přiřazený **partition** klíč
 - určuje jejich umístění v clusteru
 - může být i složený z více sloupců (composite)
 - všechna data v partition uložena spolu
 - data v partition reprezentována jako jeden nebo více **řádků**
 - každý řádek jednoznačně identifikovaný primárním klíčem
 - může být i složený



<https://app.pluralsight.com/library/courses/cassandra-developers>

OPAKOVÁNÍ

- Cassandra prakticky
 - monitorování
 - nodetool status, nodetool ring, ...
 - dotazování pomocí Cassandra Query Language (CQL)
 - dotazovací jazyk podobný SQL
 - nemá veškeré možnosti jako SQL
 - z důvodu distribuované povahy Cassandry

```
CREATE KEYSPACE vehicle_tracker
WITH REPLICATION = { 'class' : 'SimpleStrategy',
'replication_factor' : 1};
```

```
CREATE TABLE activity(
    home_id text,
    datetime timestamp,
    event text,
    code_used text,
    PRIMARY KEY (home_id, datetime)
) WITH CLUSTERING ORDER BY (datetime DESC);
```

```
INSERT INTO activity (home_id, datetime, event,
code_used) VALUES ('H01474777', '2014-05-21 07:32:16',
'alarm set', '5599');
```

```
SELECT home_id, datetime, event FROM activity;
```

- primární klíč, partition klíč, clustering sloupce
 - z jednoduchého primárního klíče je partition klíč celý primární klíč
 - ze složeného primárního klíče je partition klíč první část primárního klíče

```
CREATE TABLE activity (
    home_id text,
    datetime timestamp,
    event text,
    code_used text,
    PRIMARY KEY (home_id, datetime)
)
```


ČÁST II.: CASSANDRA



VKLÁDÁNÍ PRAKTICKY

- vkládání dat
 - INSERT INTO <jméno tabulky> (<sloupec1>, <sloupec2>, ...) VALUES (<hodnota1>, <hodnota2>, ...) USING <možnosti>;
 - vhodné pro zápis jednoho záznamu z CQLSH nebo z klientské aplikace

```
INSERT INTO activity (home_id, datetime, event,
code_used) VALUES ('H01474777', '2014-05-21 07:32:16',
'alarm set', '5599');
```

<https://www.udemy.com/course/apache-cassandra>

```
cqlsh:vehicle_tracker> INSERT INTO activity (home_id, datetime, event, code_used)
VALUES ('H01474777', '2014-05-21 07:32:16', 'alarm set', '5599');
cqlsh:vehicle_tracker> SELECT * FROM activity;

 home_id | datetime           | code_used | event
-----+-----+-----+-----+
 H01474777 | 2014-05-21 05:32:16.000000+0000 |      5599 | alarm set

(1 rows)
```

- nemusí být uvedeny hodnoty pro všechny sloupce

```
cqlsh:vehicle_tracker> INSERT INTO activity (home_id, datetime, event)
VALUES ('H01474777', '2014-05-21 08:32:16', 'alarm');
cqlsh:vehicle_tracker> SELECT * FROM activity;

 home_id | datetime           | code_used | event
-----+-----+-----+-----+
 H01474777 | 2014-05-21 06:32:16.000000+0000 |      null |     alarm
 H01474777 | 2014-05-21 05:32:16.000000+0000 |      5599 | alarm set
```



VKLÁDÁNÍ PRAKTICKY

- vkládání dat
 - INSERT INTO <jméno tabulky> (<sloupec1>, <sloupec2>, ...) VALUES (<hodnota1>, <hodnota2>, ...);
 - podpora také pro načítání z JSON formátu

```
INSERT INTO users (id, age, state)
VALUES ('user123', 42, 'TX');[/java]
```

```
INSERT INTO users
JSON '{"id": "user123", "age": 42, "state": "TX"}';
```

<https://jaxenter.com/whats-new-cassandra-3-0-122855.html>

```
cqlsh> INSERT INTO cycling.cyclist_category JSON'{
  "category" : "GC",
  "points" : 780,
  "id" : "829aa84a-4bba-411f-a4fb-38167a987cda",
  "lastname" : "SUTHERLAND" }';
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useInsertJSON.html



VKLÁDÁNÍ PRAKTICKY

- vkládání dat do kolekcí
 - INSERT INTO
 - set

```
cqlsh>INSERT INTO cycling.cyclist_career_teams (id,lastname,teams)  
VALUES (5b6962dd-3f90-4c93-8f61-eabfa4a803e2, 'VOS',  
{ 'Rabobank-Liv Woman Cycling Team', 'Rabobank-Liv Giant', 'Rabobank Women Team', 'Nederland bloeit' } );
```

- list
- map
 - složené závorky, hodnoty oddělené dvojtečkou

```
cqlsh> INSERT INTO cycling.cyclist_teams (id, lastname, firstname, teams)  
VALUES (  
5b6962dd-3f90-4c93-8f61-eabfa4a803e2,  
'VOS',  
'Marianne',  
{2015 : 'Rabobank-Liv Woman Cycling Team', 2014 : 'Rabobank-Liv Woman Cycling Team', 2013 : 'Rabobank-Liv Giant',  
2012 : 'Rabobank Women Team', 2011 : 'Nederland bloeit' } );
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useInsertDataTOC.html



VKLÁDÁNÍ PRAKTICKY

- vkládání dat do tuple
 - INSERT INTO

```
cqlsh> INSERT INTO cycling.nation_rank (nation, info) VALUES ('Spain', (1,'Alejandro VALVERDE' , 9054));
```

```
cqlsh> INSERT INTO cycling.popular (rank, cinfo) VALUES (4, ('Italy', 'Fabio ARU', 163));
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useInsertTuple.html

- analogicky i vkládání do vlastních definovaných datových typů



VKLÁDÁNÍ PRAKTIKY

- vkládání dat
 - COPY
 - slouží pro
 - import dat z CSV (COPY FROM)

```
1 home_id|datetime|event|code_used
2 H02257222|2014-05-21 05:29:47|alarm set|1566
3 H01474777|2014-05-21 07:32:16|alarm set|5599
4 H01033638|2014-05-21 07:50:22|alarm set|2121
5 H01033638|2014-05-21 07:50:43|alarm turned off|2121
6 H01033638|2014-05-21 07:55:58|alarm set|2121
7 H01545551|2014-05-21 08:30:14|alarm set|8889
8 H00999943|2014-05-21 09:05:54|alarm set|1245
9 H01033638|2014-05-21 13:02:11|alarm turned off|1919
```

```
cqlsh:vehicle_tracker> COPY activity (home_id, datetime, event, code_used)
    FROM 'D:\events.csv' WITH header = true AND delimiter = '|';
```

- potřeba specifikovat tabulku a sloupce, do kterých bude proveden import
- zdrojový soubor (FROM)
- jestli soubor obsahuje hlavičku (header)
- použitý oddělovač

- export dat do CSV (COPY TO)

```
cqlsh:vehicle_tracker> SELECT * FROM activity;
```

home_id	datetime	code_used	event
H01474777	2014-05-23 19:28:41.000000+0000	5599	alarm turned off
H01474777	2014-05-23 08:44:23.000000+0000	5599	alarm set
H01474777	2014-05-22 18:22:15.000000+0000	5599	alarm turned off
H01474777	2014-05-22 12:44:07.000000+0000	null	alarm reset by office
H01474777	2014-05-22 12:25:00.000000+0000	null	police called
H01474777	2014-05-22 12:23:59.000000+0000	null	alarm breached
H01474777	2014-05-22 08:44:13.000000+0000	5599	alarm set
H01474777	2014-05-21 19:30:33.000000+0000	5599	alarm turned off
H01474777	2014-05-21 08:32:16.000000+0000	5599	alarm set
H01474777	2014-05-21 06:32:16.000000+0000	null	alarm



VKLÁDÁNÍ DAT

- jak ale vkládání dat funguje?
 - v CQL je jednoznačným identifikátorem řádku primární klíč

home_id	datetime	event	code_used
H01474777	2014-05-22 07:44:13	alarm set	5599
H01474777	2014-05-21 18:30:33	alarm turned off	5599
H01474777	2014-05-21 07:32:16	alarm set	5599

<https://www.udemy.com/course/apache-cassandra>

- interně je to ale partition key

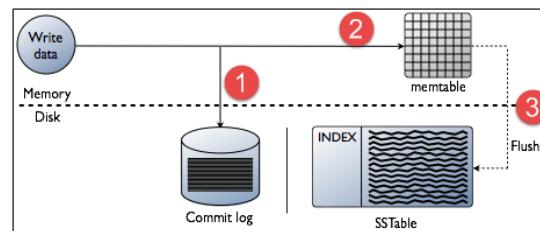
H01474777	2014-05-22 07:44:13 alarm set 5599	2014-05-21 18:30:33 alarm turned off 5599	2014-05-21 07:32:16 alarm set 5599
-----------	--	---	--

- interně jsou tedy horní tři řádky uloženy v jedné partition
- buňky + struktura řádků



UKLÁDÁNÍ DAT

- jak jsou data ukládána na disk?
 - při zápisu dat do tabulky jsou data zapsána do commit logu a do paměti
 - commit log
 - uložený na disku
 - slouží pro obnovu dat v případě selhání uzlu (durability)
 - po zápisu do SSTables archivován, smazán nebo recyklován
 - paměť (memtable)
 - mezipaměť zpětného zápisu
 - jakmile je paměť naplněna, je zapsána na disk jako SSTables
 - existuje pro každou tabulku na každém uzlu



<https://www.guru99.com/cassandra-architecture.html>



UKLÁDÁNÍ DAT

- jak jsou data ukládána na disk?
 - SSTable
 - setříděná string tabulka uložená na disku
 - umístění specifikováno v konfiguračním souboru cassandra.yaml
 - obsah je možné zobrazit pomocí .\tools\bin\sstabledump
 - vynutit zápis z memcache do SSTable je možné přes .\bin\nodetool flush <jméno>

```
(base) PS D:\Programy\apache-cassandra-3.11.9> ls .\data\data\vehicle_tracker\activity-4ae5034054d211eba88de3cb1b019d89\  
  
Directory: D:\Programy\apache-cassandra-3.11.9\data\data\vehicle_tracker\activity-4ae5034054d211eba88de3cb1b019d89
```

Mode	LastWriteTime	Length	Name
d----		-----	
-a---	12.1.2021	13:32	backups
-a---	15.1.2021	11:43	43 md-1-big-CompressionInfo.db
-a---	15.1.2021	11:43	601 md-1-big-Data.db
-a---	15.1.2021	11:43	10 md-1-big-Digest crc32
-a---	15.1.2021	11:43	24 md-1-big-Filter.db
-a---	15.1.2021	11:43	69 md-1-big-Index.db
-a---	15.1.2021	11:43	4866 md-1-big-Statistics.db
-a---	15.1.2021	11:43	71 md-1-big-Summary.db
-a---	15.1.2021	11:43	100 md-1-big-TOC.txt

```
(base) PS D:\Programy\apache-cassandra-3.11.9> .\tools\bin\sstabledump activity-4ae5034054d211eba88de3cb1b019d89\md-1-big-Data.db  
  
[{"partition": {"key": [ "H01474777" ], "position": 0}, "rows": [{"type": "row", "position": 23, "clustering": [ "2014-05-23 19:28:41.000Z" ], "liveness_info": { "tstamp": "2021-01-12T13:50:45.891Z" }, "cells": [ { "name": "code_used", "value": "5599" }, { "name": "event", "value": "alarm turned off" } ]}, {"type": "row", "position": 64, "clustering": [ "2014-05-23 08:44:23.000Z" ], "liveness_info": { "tstamp": "2021-01-12T13:50:45.890Z" }, "cells": [ { "name": "code_used", "value": "5599" }, { "name": "event", "value": "alarm set" } ]}]}
```



ČTENÍ DAT PRAKTICKY

- čtení celé tabulky
 - SELECT * FROM <jméno tabulky>;

```
cqlsh> SELECT * FROM cycling.cyclist_category;
```

race_year	race_name	rank	cyclist_name
2014	4th Tour of Beijing	1	Philippe GILBERT
2014	4th Tour of Beijing	2	Daniel MARTIN
2014	4th Tour of Beijing	3	Johan Esteban CHAVES
2015	Giro d'Italia - Stage 11 - Forli > Imola	1	Ilnur ZAKARIN
2015	Giro d'Italia - Stage 11 - Forli > Imola	2	Carlos BETANCUR
2015	Tour of Japan - Stage 4 - Minami > Shinshu	1	Benjamin PRADES
2015	Tour of Japan - Stage 4 - Minami > Shinshu	2	Adam PHELAN
2015	Tour of Japan - Stage 4 - Minami > Shinshu	3	Thomas LEBAS

- zobrazení vybraných sloupců
 - SELECT <sloupce> FROM <jméno tabulky>;

```
cqlsh> SELECT category, points, lastname FROM cycling.cyclist_category;
```

- limit výpisu
 - SELECT <sloupce> FROM <jméno tabulky> LIMIT <počet>;

```
cqlsh> SELECT * From cycling.cyclist_name LIMIT 3;
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useQueryColumnsSort.html



ČTENÍ DAT PRAKTICKY

- kdy byla data do databáze vložena?
 - WRITETIME
 - vrací čas v Unix formátu

```
SELECT id, WRITETIME(author) FROM pluralsight.courses;
```



Unix time (e.g. 1430825689)

- přiřazený token?
 - TOKEN

```
cqlsh:pluralsight> select id, token(id) from courses;
```

<code>id</code>	<code>system.token(id)</code>
advanced-python	-8770044980969467119
advanced-javascript	-8036343102284305037
nodejs-big-picture	1077068953024538991
raspberry-pi-for-developers	1510057717685901069
react-big-picture	6206829246914331914

<https://app.pluralsight.com/library/courses/cassandra-developers>



ČTENÍ DAT PRAKTICKY

- čtení s podmínkou (WHERE)
 - data jsou organizována podle partition key
 - WHERE klauzule zpravidla musí obsahovat partition key
 - nebo clustering columns z primárního klíče
 - SELECT <sloupce> FROM <jméno tabulky> WHERE <podmínka>;
 - bez WHERE klauzule jsou dotazy poslány na všechny uzly

```
cqlsh:home_security> SELECT * FROM home WHERE home_id = 'H01474777';
home_id | address          | alt_phone | city        | contact_name | email           | guest_code | main_code | phone      | phone_password | state | zip
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
H01474777 | 518 Crestview Drive | null      | Beverly Hills | Jed Clampett | jclampett@bhb.com | 7778       | 5599       | 310-775-4011 | oil           | CA    | 90046

cqlsh:home_security> SELECT * FROM activity WHERE home_id = 'H01474777' AND datetime > '2014-05-22 00:00:00';
home_id | datetime          | code_used | event
-----+-----+-----+-----+
H01474777 | 2014-05-23 19:28:41.000000+0000 | 5599       | alarm turned off
H01474777 | 2014-05-23 08:44:23.000000+0000 | 5599       | alarm set
H01474777 | 2014-05-22 18:22:15.000000+0000 | 5599       | alarm turned off
H01474777 | 2014-05-22 12:44:07.000000+0000 | null       | alarm reset by office
H01474777 | 2014-05-22 12:25:00.000000+0000 | null       | police called
H01474777 | 2014-05-22 12:23:59.000000+0000 | null       | alarm breached
H01474777 | 2014-05-22 08:44:13.000000+0000 | 5599       | alarm set

```

PRIMARY KEY (home_id, datetime)

```
cqlsh:home_security> SELECT * FROM activity WHERE home_id = 'H01474777' AND code_used = '5599';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```



ČTENÍ DAT PRAKTICKY

- čtení s podmínkou (WHERE)
 - WHERE klauzule z pravidla musí obsahovat partition key
 - nebo clustering columns z primárního klíče
 - pokud neobsahuje, dotaz bude neefektivní
 - načtení všech dat z tabulky
 - vyfiltrování záznamů, které odpovídají dotazu
 - Cassandra dotaz provede jen s klauzulí ALLOW FILTERING
 - zpravidla nevhodné
 - ve starších verzích nebylo ani možné provést

```
cqlsh:home_security> SELECT * FROM activity WHERE home_id = 'H01474777' AND code_used = '5599';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpre
ictability, use ALLOW FILTERING"
```

```
cqlsh:home_security> SELECT * FROM activity WHERE home_id = 'H01474777' AND code_used = '5599' ALLOW FILTERING;
   home_id |      datetime           | code_used | event
   :-----+-----+-----+-----+
H01474777 | 2014-05-23 19:28:41.000000+0000 |     5599 | alarm turned off
H01474777 | 2014-05-23 08:44:23.000000+0000 |     5599 |      alarm set
H01474777 | 2014-05-22 18:22:15.000000+0000 |     5599 | alarm turned off
H01474777 | 2014-05-22 08:44:13.000000+0000 |     5599 |      alarm set
H01474777 | 2014-05-21 19:30:33.000000+0000 |     5599 | alarm turned off
H01474777 | 2014-05-21 08:32:16.000000+0000 |     5599 |      alarm set
```

- řešením je definování sekundárních indexů

ČTENÍ DAT PRAKTICKY

- řazení výsledků
 - SELECT <sloupce> FROM <jméno tabulky> WHERE <podmínka> ORDER BY <sloupec> <TYP>;

```
cqlsh> CREATE TABLE cycling.cyclist_cat_pts ( category text, points int, id UUID, lastname text, PRIMARY KEY (category, points) )
SELECT * FROM cycling.cyclist_cat_pts WHERE category = 'GC' ORDER BY points ASC;
```

- ve WHERE klauzuli musí být partition key
- ORDER BY vyžaduje clustering column pro setřídění

category	points	id	lastname
GC	780	829aa84a-4bba-411f-a4fb-38167a987cda	SUTHERLAND
GC	1269	220844bf-4860-49d6-9a4b-6b5d3a79cbfb	TIRALONGO

- přejmenování sloupců pro výpis

- AS

```
cqlsh> SELECT race_name, point_id, lat_long AS CITY_LATITUDE_LONGITUDE FROM cycling.route;
```

race_name	point_id	city_latitude_longitude
47th Tour du Pays de Vaud	1	('Onnens', (46.8444, 6.6667))
47th Tour du Pays de Vaud	2	('Champagne', (46.833, 6.65))
47th Tour du Pays de Vaud	3	('Novalle', (46.833, 6.6))

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useQueryColumnsSort.html



ČTENÍ DAT PRAKTICKY

- čtení složitějších datových typů
 - tuple je přečten celý
 - kolekce se také neliší od ostatních sloupců
 - čtení dat uložených v set

race_name	point_id	city_latitude_longitude
47th Tour du Pays de Vaud	1	('Onnens', (46.8444, 6.6667))
47th Tour du Pays de Vaud	2	('Champagne', (46.833, 6.65))
47th Tour du Pays de Vaud	3	('Novale', (46.833, 6.6))

```
cqlsh> SELECT lastname, teams FROM cycling.cyclist_career_teams WHERE id = 5b6962dd-3f90-4c93-8f61-eabfa4a803e2;
```

lastname	teams
VOS	{'Nederland bloeft', 'Rabobank Women Team', 'Rabobank-Liv Giant', 'Rabobank-Liv Woman Cycling Team'}

- čtení dat uložených v list

```
cqlsh> SELECT * FROM cycling.upcoming_calendar WHERE year=2015 AND month=06;
```

year	month	events
2015	6	['The Parx Casino Philly Cycling Classic', 'Critérium du Dauphiné', 'Vuelta Ciclista a Venezuela']

- čtení dat uložených v map

```
cqlsh> SELECT lastname, firstname, teams FROM cycling.cyclist_teams WHERE id=5b6962dd-3f90-4c93-8f61-eabfa4a803e2;
```

lastname	firstname	teams
VOS	Marianne	{2011: 'Nederland bloeft', 2012: 'Rabobank Women Team', 2013: 'Rabobank-Liv Giant', 2014: 'Rabobank-Liv Woman Cycling Team', 2015: 'Rabobank-Liv Woman Cycling Team'}

- pro čtení části kolekce je nutné definovat sekundární index



ČTENÍ DAT PRAKTICKY

- čtení v JSON
 - klíčové slovo JSON

```
cqlsh:cycling> select json name, checkin_id, timestamp from checkin;  
[json]  
-----  
{"name": "BRAND", "checkin_id": "50554d6e-29bb-11e5-b345-feff8194dc9f", "timestamp": "2016-08-28  
 {"name": "VOSS", "checkin_id": "50554d6e-29bb-11e5-b345-feff819cdc9f", "timestamp": "2016-08-28  
(2 rows)
```

- pro vybrané sloupce funkce toJson()

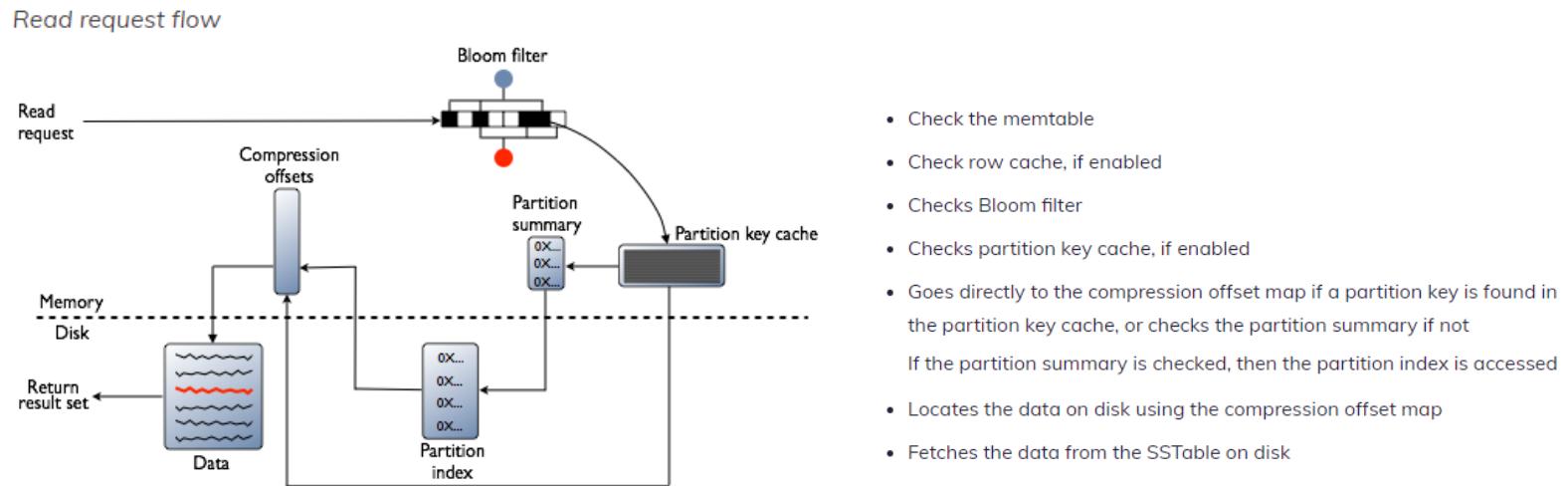
```
cqlsh:cycling> select name, checkin_id, toJson(timestamp) from checkin;  
  
name | checkin_id | system.toJson(timestamp)  
-----+-----+-----  
BRAND | 50554d6e-29bb-11e5-b345-feff8194dc9f | "2016-08-28 21:45:10.406Z"  
VOSS | 50554d6e-29bb-11e5-b345-feff819cdc9f | "2016-08-28 21:44:04.113Z"
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useQueryJSON.html



ČTENÍ DAT

- jak je čtení dat prováděno?
 - Cassandra musí kombinovat výsledky z aktuální memtable a z SSTables
 - zpracování v několika fázích



<https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlAboutReads.html>



MODELOVÁNÍ DAT

- co je potřeba si uvědomit?
 - Cassandra je distribuovaná databáze
 - neexistují joiny
 - veškerá data k danému dotazu musí být jen v jedné tabulce
 - vede k denormalizovaným datům
 - zápisy jsou rychlé, redundancy není až takový problém
 - je tedy potřeba si dobře rozmyslet, které dotazy budou potřeba
 - a následně podle nich navrhnut tabulky
 - partition key, clustering columns a sekundární indexy
 - kontrast oproti klasickým datovým přístupům v relačních databázích

MODELOVÁNÍ DAT

- sekundární indexy
 - možnost definovat nad vybranými sloupcí
 - kromě partition key, který už indexovaný je
 - umožňují filtrování nad danými sloupcí
 - použít WHERE klauzule
 - pro každý sekundární index je vytvořena skrytá tabulka na každém uzlu
 - pro přístup
 - nezrychlují prohledávání, ale umožňují jej i nad ostatními sloupcí
 - pro zrychlení je možnost vytvořit tabulku přímo odpovídající dotazu
 - CREATE INDEX <jméno indexu> ON <jméno tabulky> (<jméno sloupce>)

MODELOVÁNÍ DAT

- sekundární indexy
 - vytvoření nad libovolným sloupcem
 - CREATE INDEX <jméno indexu> ON <jméno tabulky> (<jméno sloupce>)
 - po vytvoření použití automatické

```
cqlsh:home_security> SELECT * FROM activity WHERE code_used = '5599';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh:home_security> CREATE INDEX code_used_index ON activity (code_used);
cqlsh:home_security> SELECT * FROM activity WHERE code_used = '5599';

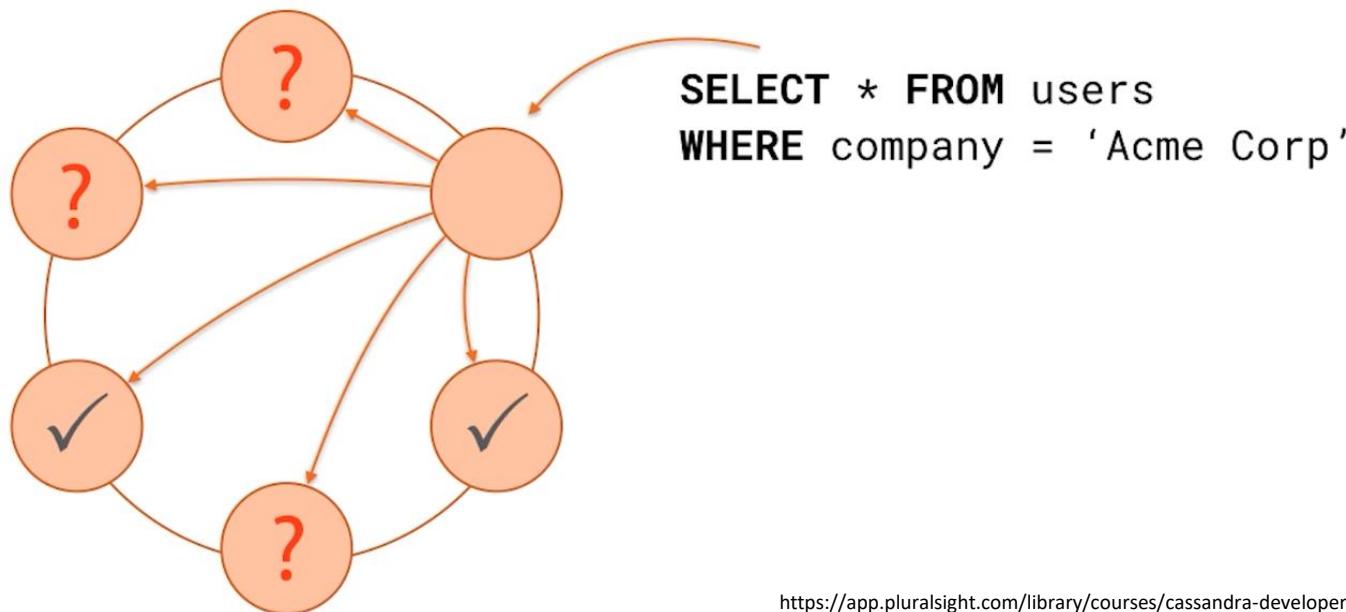
 home_id | datetime           | code_used | event
-----+-----+-----+-----+
H01474777 | 2014-05-23 19:28:41.000000+0000 |      5599 | alarm turned off
H01474777 | 2014-05-23 08:44:23.000000+0000 |      5599 | alarm set
H01474777 | 2014-05-22 18:22:15.000000+0000 |      5599 | alarm turned off
H01474777 | 2014-05-22 08:44:13.000000+0000 |      5599 | alarm set
H01474777 | 2014-05-21 19:30:33.000000+0000 |      5599 | alarm turned off
H01474777 | 2014-05-21 08:32:16.000000+0000 |      5599 | alarm set

(6 rows)
```



MODELOVÁNÍ DAT

- sekundární indexy



<https://app.pluralsight.com/library/courses/cassandra-developers>

MODELOVÁNÍ DAT

- sekundární indexy
 - vytvoření nad kolekcemi
 - set a list
 - definování indexu nad daným sloupcem (zůstává stejné)
 - př. výpis cyklistů reprezentujících tým Nederland bloeit

```
CREATE INDEX team_idx ON cycling.cyclist_career_teams ( teams );
SELECT * FROM cycling.cyclist_career_teams WHERE teams CONTAINS 'Nederland bloeit';
```

id	lastname	teams
5b6962dd-3f90-4c93-8f61-eabfa4a803e2	VOS	{'Nederland bloeit', 'Rabobank Women Team', 'Rabobank-Liv Giant', 'Rabobank-Liv Woman Cycling Team'}

- map
 - definování indexu nad klíčem, hodnotou nebo záznamem
 - př. definování nad klíčem

```
CREATE INDEX team_year_idx ON cycling.cyclist_teams ( KEYS (teams) );
SELECT * From cycling.cyclist_teams WHERE teams CONTAINS KEY 2015;
```

id	firstname	lastname	teams
cb07baad-eac8-4f65-b28a-bddc06a0de23	Elizabeth	ARMITSTEAD	{2011: 'Team Garmin - Cervelo', 2012: 'AA Drink - Leontien.nl', 2013: 'Boels-Dolmans Cycling Team', 2014: 'Boels-Dolmans Cycling Team', 2015: 'Boels-Dolmans Cycling Team'}
5b6962dd-3f90-4c93-8f61-eabfa4a803e2	Marianne	VOS	{2011: 'Nederland bloeit', 2012: 'Rabobank Women Team', 2013: 'Rabobank-Liv Giant', 2014: 'Rabobank-Liv Woman Cycling Team', 2015: 'Rabobank-Liv Woman Cycling Team'}

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useIndexColl.html



MODELOVÁNÍ DAT

- sekundární indexy
 - vytvoření nad kolekcemi
 - map
 - definování indexu nad klíčem, hodnotou nebo záznamem
 - př. definování nad hodnotou

```
);CREATE INDEX blist_idx ON cycling.birthday_list (VALUES(blist));
SELECT * FROM cycling.birthday_list CONTAINS 'NETHERLANDS';
```

cyclist_name	blist
Luc HAGENAARS	{'age': '28', 'bday': '27/07/1987', 'nation': 'NETHERLANDS'}
Toine POELS	{'age': '52', 'bday': '27/07/1963', 'nation': 'NETHERLANDS'}

- př. definování nad záznamem

```
CREATE INDEX blist_idx ON cycling.birthday_list (ENTRIES(blist));
SELECT * FROM cycling.birthday_list WHERE blist['age'] = '23';
```

cyclist_name	blist
Claudio HEINEN	{'age': '23', 'bday': '27/07/1992', 'nation': 'GERMANY'}
Laurence BOURQUE	{'age': '23', 'bday': '27/07/1992', 'nation': 'CANADA'}

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useIndexColl.html



UPDATE DAT PRAKTICKY

- příkaz UPDATE
 - UPDATE <jméno tabulky> SET <jméno sloupce – hodnota> WHERE <podmínka>
 - WHERE klauzule je volitelná

```
UPDATE home_security.home SET phone = '310-883-7197'  
WHERE home_id = 'H01474777';
```

- možnost aktualizovat více hodnot najednou
 - oddělení čárkou

```
UPDATE home_security.home SET phone = '310-883-7197',  
contact_name = 'Mr. Drysdale' WHERE home_id = 'H01474777';
```

<https://www.udemy.com/course/apache-cassandra>

```
cqlsh:home_security> SELECT home_id, address, phone, contact_name from HOME WHERE home_id = 'H01474777';  
home_id | address | phone | contact_name  
-----+-----+-----+-----  
H01474777 | 518 Crestview Drive | 310-775-4011 | Jed Clampett
```

```
cqlsh:home_security> UPDATE home SET phone = '310-883-7197' WHERE home_id = 'H01474777';
```

```
cqlsh:home_security> SELECT home_id, address, phone, contact_name from HOME WHERE home_id = 'H01474777';  
home_id | address | phone | contact_name  
-----+-----+-----+-----  
H01474777 | 518 Crestview Drive | 310-883-7197 | Jed Clampett
```





UPDATE DAT PRAKTICKY

- jak ale update probíhá?
 - situace je odlišná od relačních databází
 - neplatí vyhledání záznamu na disku, jeho aktualizace a uložení
 - update je v podstatě další zápis
 - zápis do memtable
 - po naplnění memtable uložení do nové SSTable
 - konflikty jsou řešeny pomocí časových značek
 - při čtení Cassandra prochází všechny memtables a SSTables
 - vrácený záznam je ten s poslední časovou značkou



UPDATE DAT PRAKTICKY

- update kolekcí

- set

- přidání prvku do setu pomocí +

```
cqlsh> UPDATE cycling.cyclist_career_teams
      SET teams = teams + {'Team DSB - Ballast Nedam'} WHERE id = 5b6962dd-3f90-4c93-8f61-eabfa4a803e2;
```

- odebrání pomocí -

```
cqlsh> UPDATE cycling.cyclist_career_teams
      SET teams = teams - {'WOMBATS - Womens Mountain Bike & Tea Society'} WHERE id = 5b6962dd-3f90-4c93-8f61-eabfa4a803e2;
```

- list

- přidání prvku na začátek

```
cqlsh> UPDATE cycling.upcoming_calendar SET events = ['The Parx Casino Philly Cycling Classic'] + events WHERE year = 2015
```

- přidání prvku na konec

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useInsertDataTOC.html

```
cqlsh> UPDATE cycling.upcoming_calendar SET events = events + ['Tour de France Stage 10'] WHERE year = 2015
```

- v obou případech nedochází interně ke čtení seznamu, ale jen uložení nového prvku



UPDATE DAT PRAKTICKY

- update kolekcí
 - list
 - přidání prvku na konkrétní pozici

```
cqlsh> UPDATE cycling.upcoming_calendar SET events[2] = 'Vuelta Ciclista a Venezuela' WHERE year = 2015
```

- v tomto případě Cassandra přečte celý list a přepíše prvky, u kterých se mění index
- může být velmi pomalé
- odebrání prvku

```
cqlsh> UPDATE cycling.upcoming_calendar SET events = events - ['Tour de France Stage 10'] WHERE year = 2015
```

- map
 - přidání prvku do map

```
cqlsh> UPDATE cycling.cyclist_teams SET teams = teams + {2009 : 'DSB Bank - Nederland bloeit'} WHERE id = 5b6962dd-3f90-4c93
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useInsertDataTOC.html



UPDATE DAT PRAKTICKY

- update kolekcí

- map
 - změna hodnoty prvku

```
cqlsh> UPDATE cycling.cyclist_teams SET teams[2006] = 'Team DSB - Ballast Nedam' WHERE id = 5b6962dd-3f90-4c93
```

- odebrání prvku z mapy

```
cqlsh> UPDATE cycling.cyclist_teams SET teams = teams - {'2013','2014'} WHERE id=e7cd5752-bc0d-4157
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useInsertDataTOC.html

- update je možný i na uživatelem definované datové typy



ODSTRANĚNÍ DAT PRAKTICKY

- příkazy DELETE, TRUNCATE a DROP
 - DELETE
 - odstranění hodnot ve sloupcích

```
cqlsh> DELETE lastname FROM cycling.cyclist_name WHERE id = c7fceba0-c141-4207-9494-a29f9809de6f;
```

- odstranění řádků tabulky

```
cqlsh> DELETE FROM cycling.calendar WHERE race_id = 200;
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useDropColumn.html



ODSTRANĚNÍ DAT PRAKTICKY

- příkazy DELETE, TRUNCATE a DROP
 - TRUNCATE
 - odstranění všech řádků tabulky

```
TRUNCATE cycling.user_activity;
```

```
TRUNCATE TABLE cycling.user_activity;
```

- DROP
 - odstranění tabulky nebo keyspace

```
DROP TABLE cycling.cyclist_name;
```

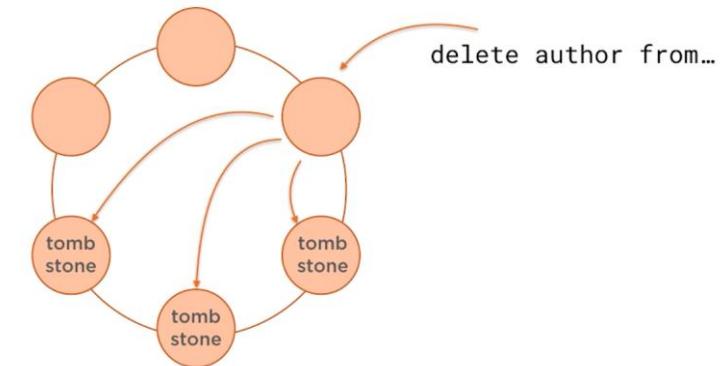
```
DROP KEYSPACE cycling;
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useDropKSTableMV.html



ODSTRANĚNÍ DAT PRAKTICKY

- co se stane při odstranění?
 - při zadání příkazu pro odstranění je vytvořen tzv. tombstone
 - ten označuje data pro smazání
 - proč ale nejsou data smazána ihned?
 - důvodem je distribuovaná povaha Cassandra
 - při okamžitém smazání by mohlo dojít ke zpětné replikaci z jiného uzlu
 - který nemusel v době smazání být k dispozici
 - uzly (i ty momentálně neběžící) tak dostávají čas, aby se o označení dozvěděly
 - doba mezi označením a možným smazáním dána ve vlastnostech tabulky
 - gc_grace_seconds
 - v základu 10 dní (864 000 sekund)
 - po uplynutí této doby je data možné smazat



<https://app.pluralsight.com/library/courses/cassandra-developers>



ODSTRANĚNÍ DAT PRAKTICKY

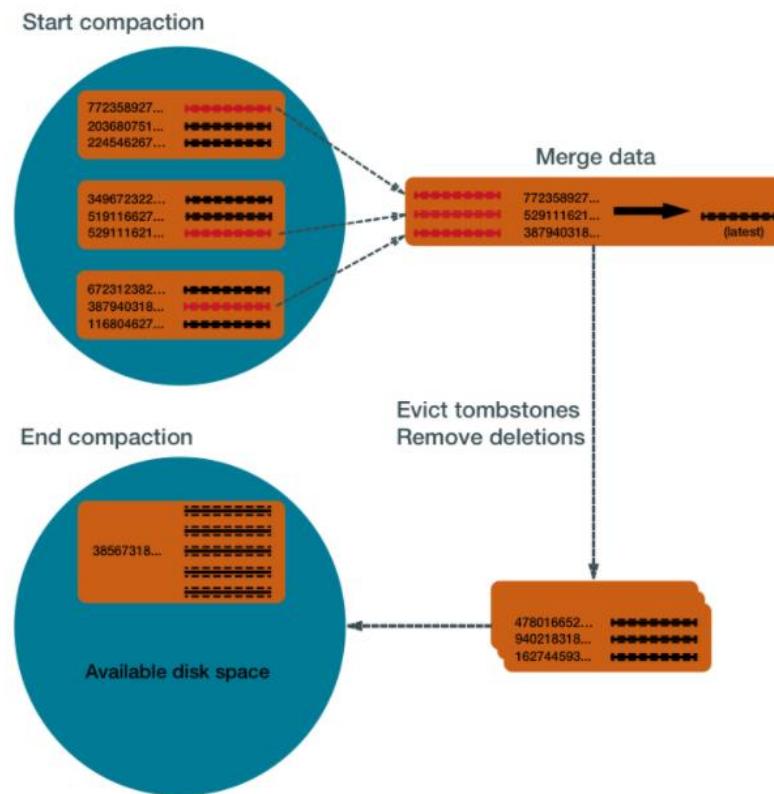
- co se stane při odstranění?
 - pro samotné smazání musí nastat compaction
 - compaction nastává při sloučení SSTables
 - zlepšení výkonu čtení z důvodu menšího počtu SSTables
 - navrácení diskové kapacity smazáním dat
 - prováděno automaticky
 - při existenci 4 velkých SSTables je vytvořena sloučením nová a původní jsou smazány
 - lze vyvolat manuálně přes .\bin\nodetool compact

```
CREATE TABLE home_security.activity (
    home_id text,
    datetime timestamp,
    code_used text,
    event text,
    PRIMARY KEY (home_id, datetime)
) WITH CLUSTERING ORDER BY (datetime DESC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND delocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
CREATE INDEX code_used_index ON home_security.activity (code_used);
```



ODSTRANĚNÍ DAT PRAKTICKY

- co se stane při odstranění?



<https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataMaintain.html>

TIME TO LIVE

- time to live (TTL)
 - způsob, kterým lze vkládaným datům nastavit expirační lhůtu
 - pomocí USING TTL

```
INSERT INTO location (vehicle_id, date, time, latitude,  
longitude) VALUES ('AZWT3RSKI', '2014-05-20', '2014-05-20  
11:23:55', 34.872689, -111.757373) USING TTL 30;
```

<https://www.udemy.com/course/apache-cassandra>

- v sekundách (např. 30 vteřin)
- po vypršení TTL nastává standardní procedura pro smazání
 - tombstone
 - gc_grace_period
 - compaction



TIME TO LIVE

- time to live (TTL)
 - hodnotu TTL lze upravit pomocí příkazu UPDATE

```
UPDATE location USING TTL 7776000 SET latitude = 34.872689,  
longitude = -111.757373 WHERE vehicle_id = 'AZWT3RSKI' AND  
date = '2014-05-20' AND time='2014-05-20 11:23:55';
```

<https://www.udemy.com/course/apache-cassandra>

- zobrazení TTL

```
SELECT TTL (race_name) from cycling.calendar WHERE race_id = 200;
```

ttl(race_name)
86371



STATICKÉ SLOUPCE

- sloupce, jejichž hodnoty jsou konstantní pro všechny řádky v partition
- STATIC

```
CREATE TABLE courses (
    id varchar,
    name varchar STATIC,
    module_id int,
    module_name varchar,
    PRIMARY KEY (id, module_id)
);
```

module_id=1	
	name Advanced Python
	module_name Course Overview
module_id=2	
	name Advanced Python
	module_name Advanced Flow Control

name	Advanced Python
module_id=1	
module_name	Course Overview
module_id=2	
	module_name Advanced Flow Control

<https://app.pluralsight.com/library/courses/cassandra-developers>



POČÍTADLA

- speciální datový typ (counter)
 - podporující inkrementaci o integer
- vyžadují speciální péči v distribuovaném prostředí
 - speciální sloupec tabulky
 - nemohou být primárním klíčem
 - nemohou být indexovány
 - aktualizace přes update
 - jen dedikované tabulky
 - obsahují jen primární klíč a counter(y)

```
CREATE TABLE pluralsight.ratings (
    course_id varchar PRIMARY KEY,
    ratings_count counter,
    ratings_total counter
);
```

```
UPDATE pluralsight.ratings
SET ratings_count = ratings_count + 1,
    ratings_total = ratings_total + 4
WHERE course_id = 'cassandra-developers';
```

<https://app.pluralsight.com/library/courses/cassandra-developers>



AGREGAČNÍ FUNKCE

- základní funkce jsou vestavěné
 - minimum (min)
 - maximum (max)
 - průměr (avg)
 - suma (sum)

id	race_points	firstname	lastname
e3b19ec4-774a-4d1c-9e5a-decec1e30aac	6	Giorgia	BRONZINI
e3b19ec4-774a-4d1c-9e5a-decec1e30aac	75	Giorgia	BRONZINI
e3b19ec4-774a-4d1c-9e5a-decec1e30aac	120	Giorgia	BRONZINI

```
cqlsh> SELECT sum(race_points) FROM cycling.cyclist_points WHERE id=e3b19ec4-774a-4d1c-9e5a-decec1e30aac;
```

```
system.sum(race_points)
-----
201
```

- počet (count)
 - př. počet soutěžících z Belgie

country	cyclist_name	flag
Belgium	Andre	1
Belgium	Jacques	1
France	Andre	3
France	George	3

```
cqlsh> SELECT count(cyclist_name) FROM cycling.country_flag WHERE country='Belgium';
```

```
system.count(cyclist_name)
-----
2
```

- možnost definovat vlastní agregační funkce



FUNKCE DEFINOVANÉ UŽIVATELEM

- uživatelské funkce (UDF)
 - aplikované na data v tabulce v rámci dotazu
 - musí být vytvořeny před použití v SELECT dotazu
 - aplikovány na všechny záznamy
 - nutno povolit v cassandra.yaml
 - v základu Java nebo Javascript
 - možnost rozšířit např. o Python, Ruby nebo Scala
 - př. logaritmus

```
CREATE FUNCTION IF NOT EXISTS fLog (input double)
  CALLED ON NULL INPUT
  RETURNS double
  LANGUAGE java AS '
    return Double.valueOf(Math.log(input.doubleValue()));
  ';
https://docs.datastax.com/en/cql-oss/3.3/cql/cql\_using/useCreateUDF.html
```

- CALLED ON NULL INPUT – zajistí, že funkce bude vždy provedena
- RETURNS NULL ON NULL INPUT – vrátí pro vstupní NULL výstupní NULL
- RETURNS – definuje vrácený datový typ



FUNKCE DEFINOVANÉ UŽIVATELEM

- uživatelské funkce (UDF)
 - možnost vytvořit také s CREATE OR REPLACE FUNCTION
 - dojde k nahrazení funkce se stejným jménem, pokud existuje

```
cqlsh> CREATE OR REPLACE FUNCTION fLog
```

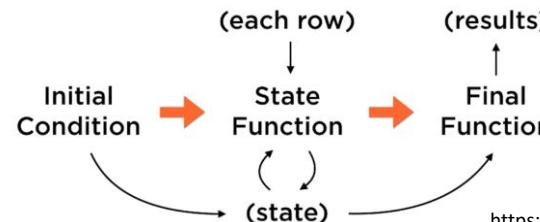
- čtení

```
cqlsh> SELECT id, lastname, fLog(race_points) FROM cycling.cyclist_points;
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useQueryUDF.html

<code>id</code>	<code>lastname</code>	<code>cycling.flog(race_points)</code>
220844bf-4860-49d6-9a4b-6b5d3a79cbfb	TIRALONGO	0.693147
e3b19ec4-774a-4d1c-9e5a-decec1e30aac	BRONZINI	1.79176
e3b19ec4-774a-4d1c-9e5a-decec1e30aac	BRONZINI	4.31749
e3b19ec4-774a-4d1c-9e5a-decec1e30aac	BRONZINI	4.78749

- možnost definovat a používat i vlastní agregační funkce



<https://app.pluralsight.com/library/courses/cassandra-developers>



ČASOVÉ ŘADY

- data z různých zdrojů
- plná podpora v Cassandře
 - datový typ TimeUUID



45b94a50-12e5-11e5-9114-091830ac5256

- UUID v1
- 3 části
 - počet 100 ns intervalů od UUID epochy
 - MAC adresa
 - clock sequence number zabraňující duplikátům
- unikátní a setříditelné podle data a času
- funkce
 - now()
 - dateOf(), unixTimestampOf()
 - minTimeuuid(), maxTimeuuid()

```
CREATE TABLE course_page_views (
    course_id text,
    view_id timeuuid,
    PRIMARY KEY (course_id, view_id)
) WITH CLUSTERING ORDER BY (view_id DESC);
```

<https://app.pluralsight.com/library/courses/cassandra-developers>

```
INSERT INTO course_page_views (course_id, view_id)
VALUES ('advanced-python', now());
```

```
SELECT dateOf(view_id)
FROM course_page_views
WHERE course_id = 'advanced-python'
AND view_id >= maxTimeuuid('2019-11-01 00:00+0000')
AND view_id < minTimeuuid('2019-12-01 00:00+0000')
```



DÁVKY

- **batches**
 - navržené pro udržení příbuzných tabulek (nebo řádků) v synchronizaci
 - ne nezbytně pro rychlé načítání dat
- **skupina CQL dotazů poslaných společně**
 - o postupném zpracovávání dotazů jsou ostatní uzly informovány
 - zajišťuje integritu (a dokončení) dávek
 - nejedná se ale o transakce
 - není rollback
 - nelogované (unlogged) dávky
 - nedochází k informování ostatních uzlů
 - vhodné pro zápis více dat do jedné partition
 - posláno jako jediný zápis



DÁVKY

BEGIN BATCH

```
INSERT INTO courses (id, tags)
VALUES ('nodejs-big-picture',
{'developer', 'javascript', 'node.js', 'open-source'});
```

```
INSERT INTO course_tags (tag, course_id)
VALUES ('developer', 'nodejs-big-picture');
```

// ... etc.

BEGIN UNLOGGED BATCH

APPLY BATCH;

```
INSERT INTO courses (id, name)
VALUES ('nodejs-big-picture', 'Node.js: The Big Picture');
```

```
INSERT INTO courses (id, module_id, module_name)
VALUES ('nodejs-big-picture', 1, 'Course Overview');
```

// ... etc.

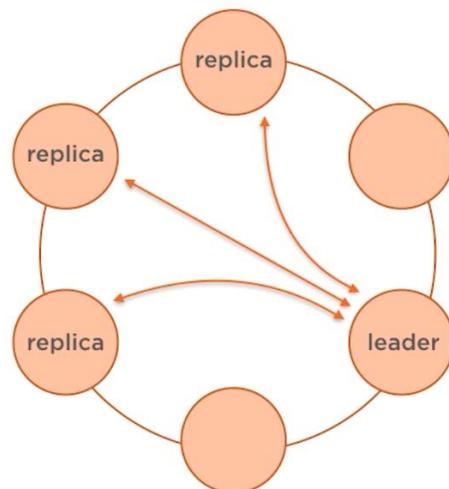
APPLY BATCH;

<https://app.pluralsight.com/library/courses/cassandra-developers>



LIGHTWEIGHT TRANSAKCE

- přidané v posledních verzích Cassandra
- implementace [Paxos](#) algoritmu
 - každá transakce obsahuje leader uzel a repliky
 - uzly spolu komunikují, aby dokončily transakci
 - 4 kroky



1. **Prepare** ↔ **Promise**
2. **Read** ↔ **Results**
3. **Propose** ↔ **Accept**
4. **Commit** ↔ **Ack**

<https://app.pluralsight.com/library/courses/cassandra-developers>



LIGHTWEIGHT TRANSAKCE

- zpřístupněné v rámci Compare-and-Set operací
 - pro INSERT dotaz IF NOT EXISTS
 - provede se jen v případě, že záznam neexistuje

Insert

```
INSERT INTO users (id, first_name, last_name)
VALUES ('john-doe', 'John', 'Doe')
IF NOT EXISTS;
```

- pro UPDATE dotaz IF
 - provede se jen v případě, že je splněná podmínka

Update

```
UPDATE users SET password = 'mypass', reset_token = null
WHERE id = 'john-doe'
IF reset_token = '1GRhEs1';
```

<https://app.pluralsight.com/library/courses/cassandra-developers>

- náročné





MATERIALIZED VIEWS

- speciální tabulka s daty, která jsou automaticky vkládána a aktualizována podle zdrojové tabulky
 - tabulky se liší primárním klíčem a vlastnostmi
 - umožňuje provádět jiné optimalizované dotazy
 - lišící se primární klíče
 - a zároveň udržuje data aktuální podle zdrojové tabulky
 - kontrast oproti řešení, kdy pro každý dotaz existuje nezávislá tabulka
 - v tomto případě je potřeba data aktualizovat ve všech tabulkách
 - vhodné pro data s velkou kardinalitou
 - sloupce obsahující co nejvíce unikátních záznamů
 - pro data s malou kardinalitou výhodnější sekundární indexy



MATERIALIZED VIEWS

- speciální tabulka s daty, která jsou automaticky vkládána a aktualizována podle zdrojové tabulky
 - příkaz CREATE MATERIALIZED VIEW

```
CREATE TABLE IF NOT EXISTS cycling.cyclist_base (
    cid UUID PRIMARY KEY,
    name text,
    age int,
    birthday date,
    country text
);
```

```
CREATE MATERIALIZED VIEW IF NOT EXISTS cycling.cyclist_by_country
AS SELECT age, birthday, name, country
FROM cycling.cyclist_base
WHERE country IS NOT NULL
AND cid IS NOT NULL
PRIMARY KEY (country, cid);
```

- AS SELECT vybírá sloupce ze zdrojové tabulky do materialized view
- FROM definuje zdrojovou tabulku
- WHERE klauzule musí obsahovat všechny sloupce primárního klíče s IS NOT NULL
 - zajišťuje, aby v primárním klíči nebyly NULL hodnoty
- primární klíč musí obsahovat všechny prvky původního primárního klíče
- nutná specifikace primárního klíče

```
SELECT age, name, birthday
FROM cycling.cyclist_by_country
WHERE country = 'Netherlands';
```

https://docs.datastax.com/en/dse/6.8/cql/cql/cql_using/useOverviewMV.html





MATERIALIZED VIEWS

```
CREATE TABLE users (
    id varchar,
    first_name varchar,
    last_name varchar,
    company varchar,
    // ...
    PRIMARY KEY (id)
);
```

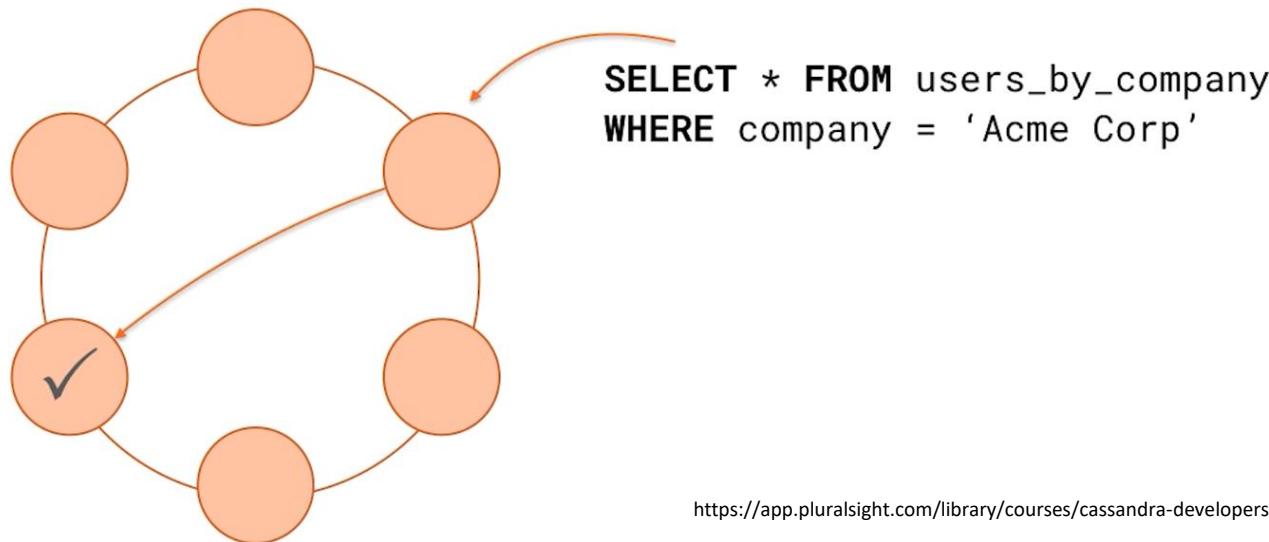
```
CREATE MATERIALIZED VIEW
users_by_company AS
SELECT * FROM users
WHERE company IS NOT null
PRIMARY KEY (company, id);

SELECT first_name, last_name
FROM users_by_company
WHERE company = 'Acme Corp'
```

<https://app.pluralsight.com/library/courses/cassandra-developers>



MATERIALIZED VIEWS



<https://app.pluralsight.com/library/courses/cassandra-developers>

KLIENTSKÉ DRIVERY

- pro komunikaci s CQL Cassandra API lze kromě cqlsh použít klientské drivery

Java

- [Achilles](#)
- [Astyanax](#)
- [Casser](#)
- [Datastax Java driver](#)
- [Kundera](#)
- [PlayORM](#)

Python

- [Datastax Python driver](#)

Ruby

- [Datastax Ruby driver](#)

C# / .NET

- [Cassandra Sharp](#)
- [Datastax C# driver](#)
- [Fluent Cassandra](#)

Nodejs

- [Datastax Nodejs driver](#)
- [Node-Cassandra-CQL](#)

PHP

- [CQL | PHP](#)
- [Datastax PHP driver](#)
- [PHP-Cassandra](#)
- [PHP Library for Cassandra](#)

C++

- [Datastax C++ driver](#)
- [libQTCassandra](#)

Scala

- [Datastax Spark connector](#)
- [Phantom](#)
- [Quill](#)

Clojure

- [Alia](#)
- [Cassaforte](#)
- [Hayt](#)

Erlang

- [CQerl](#)
- [Erlcassandra](#)

Go

- [CQLC](#)
- [Gocassa](#)
- [GoCQL](#)

Haskell

- [Cassy](#)

Rust

- [Rust CQL](#)

Perl

- [Cassandra::Client and DBD::Cassandra](#)

Elixir

- [Xandra](#)
- [CQEx](#)

Dart

- [dart_cassandra_cql](#)





KLIENTSKÉ DRIVERY - PYTHON

- Datastax Python Driver

- pář příkazů
- vytvoření clusteru

```
from cassandra.cluster import Cluster  
  
cluster = Cluster()
```

- vytvoření session pro připojení k uzlu

```
cluster = Cluster()  
session = cluster.connect()
```

- přepnutí keyspace

```
session.set_keyspace('users')  
# or you can do this instead  
session.execute('USE users')
```

<https://docs.datastax.com/en/developer/python-driver/3.24/>





KLIENTSKÉ DRIVERY - PYTHON

- Datastax Python Driver

- provádění příkazů
 - pomocí Session.execute
 - čtení dat z tabulky

```
rows = session.execute('SELECT name, age, email FROM users')
for user_row in rows:
    print user_row.name, user_row.age, user_row.email
```

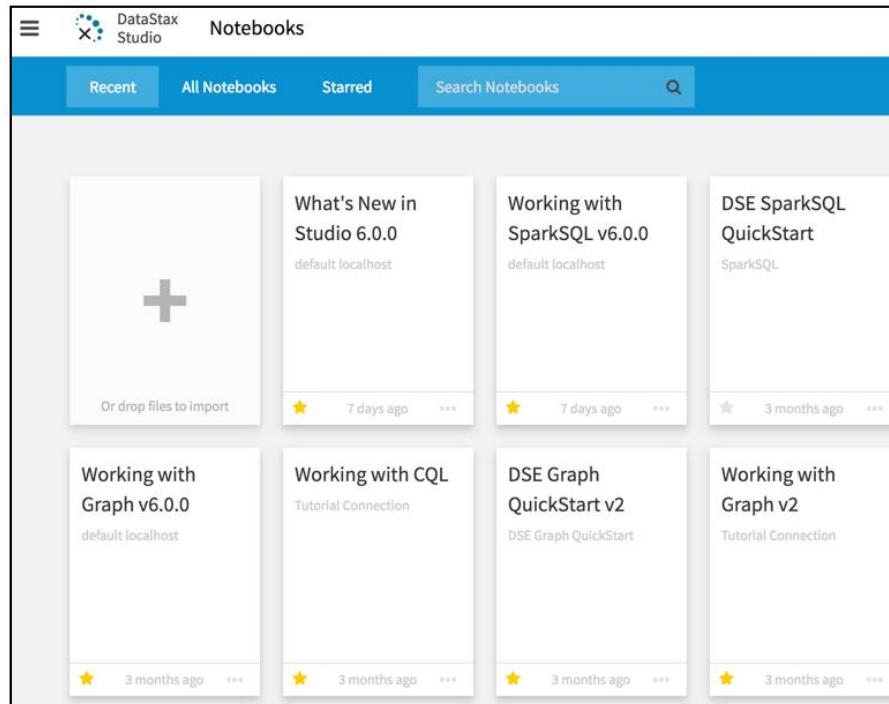
<https://docs.datastax.com/en/developer/python-driver/3.24/>

- data vrácena ve struktuře tuple
- více detailů v dokumentaci
- použití dalších driverů je velmi podobné
 - pozor na verzi podporované Cassandra



DATASTAX STUDIO

- pro komunikaci s Cassandra API lze také použít Datastax Studio



<https://docs.datastax.com/en/studio/6.0/studio/createSimpleNotebook.html>

A PŘÍŠTĚ?

- sloupcové databáze
 - Cassandra...





Děkuji za pozornost.
Otázky?

