

**B4M36DS2: Database Systems 2**

<http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-1-B4M36DS2/>

Lecture 10

# **Column-Family Stores: Cassandra**

**Martin Svoboda**

svoboda@ksi.mff.cuni.cz

5. 12. 2016

**Charles University in Prague**, Faculty of Mathematics and Physics

**Czech Technical University in Prague**, Faculty of Electrical Engineering

# Lecture Outline

## Column-family stores

- General introduction

## Apache Cassandra

- Data model
- Cassandra query language
  - DDL statements
  - DML statements

# Wide Column Stores

## Data model

- **Column family** (table)
  - Table is a collection of **similar rows** (not necessarily identical)
- **Row**
  - Row is a collection of **columns**
    - Should encompass a group of data that is accessed together
  - Associated with a unique **row key**
- **Column**
  - Column consists of a **column name** and **column value** (and possibly other metadata records)
  - Scalar values, but also **flat sets, lists or maps** may be allowed

# Wide Column Stores

## Query patterns

- Create, update or remove a row within a given column family
- **Select rows according to a row key or simple conditions**

## Warning

- Wide column stores are not just a special kind of RDBMSs with a variable set of columns!

# Wide Column Stores

## Suitable use cases

- Event logging, content management systems, blogs, ...
  - I.e. **for structured flat data with similar schema**

## When not to use

- **ACID transactions** are required
- **Complex queries:** aggregation (SUM, AVG, ...), joining, ...
- Early prototypes: i.e. when **database design may change**

## Representatives

- Apache **Cassandra**, Apache **HBase**, Apache Accumulo, Hypertable, **Google Bigtable**

# Wide Column Stores

## Representatives



# Apache Cassandra



# Apache Cassandra

## Column-family database

- <http://cassandra.apache.org/>
- Features
  - Open-source, high availability, linear scalability, sharding (spanning multiple datacenters), peer-to-peer configurable replication, tunable consistency, MapReduce support
- Developed by **Apache Software Foundation**
  - Originally at Facebook
- Implemented in Java
- Operating systems: cross-platform
- Initial release in 2008



# Data Model

## Database system structure

Instance → **keyspaces** → **tables** → **rows** → **columns**

- Keyspace
- Table (column family)
  - **Collection of (similar) rows**
  - Table schema must be specified, yet can be modified later on
- Row
  - **Collection of columns**
  - Rows in a table do not need to have the same columns
  - Each row is **uniquely identified** by a **primary key**
- Column
  - **Name-value pair** + additional data

# Data Model

## Column values

- Empty value
  - `null`
- Atomic value
  - **Native data types** such as texts, integers, dates, ...
  - **Tuples**
    - Tuple of anonymous fields, each of any type (even different)
  - **User defined types (UDT)**
    - Set of named fields of any type
- Collections
  - **Lists, sets, and maps**
    - Nested tuples, UDTs, or collections are allowed, but currently only in **frozen mode** (such elements are serialized when stored)

# Data Model

## Collections

- **List**
  - **Sorted collection of non-unique values**
  - List elements are ordered by their positions
  - Not always recommended because of performance issues
    - Internal read-before-write operations have to be executed
- **Set**
  - **Sorted collection of unique values**
- **Map**
  - **Sorted collection of key-value pairs**
  - Map elements are ordered by their keys
  - Keys must be unique

# Sample Data

Table of **actors**

id			
trojan	name	year	movies
	( Ivan, Trojan )	1964	{ samotari, medvidek }
machacek	name	year	
	( Jiří, Macháček )	1966	
	movies		
	{ medvidek, vratnelahve, samotari }		
schneiderova	name	year	movies
	( Jitka, Schneiderová )	1973	{ samotari }
sverak	name	year	movies
	( Zdeněk, Svěrák )	1936	{ vratnelahve }

# Sample Data

Table of **movies**

id				
samotari	title	year	actors	genres
	Samotáři	2000	null	[ comedy, drama ]
medvidek	title	director	year	
	Medvídek	( Jan, Hřebejk )	2007	
	actors			
	{ trojan: Ivan, machacek: Jirka }			
vratnelahve	title	year	actors	
	Vratné lahve	2006	{ machacek: Robert Landa }	
zelary	title	year	actors	genres
	Želary	2003	{ }	[ romance, drama ]

# Data Model

**Additional data** associated with...

the whole column in case of atomic values, or  
every element of a collection

- **Time-to-live (TTL)**
  - After a certain amount of time (number of seconds) a given value is automatically deleted
- **Timestamp (writetime)**
  - Timestamp of the last value modification
  - Assigned automatically or manually as well
- Both the records can be queried
  - Unfortunately not in case of collections and their elements

# Cassandra API

## CQLSH

- **Interactive command line shell**
- `bin/cqlsh`
- Uses **CQL** (*Cassandra Query Language*)

## Client drivers

- Provided by the community
- Available for various languages
  - Java, Python, Ruby, PHP, C++, Scala, Erlang, ...

# Query Language

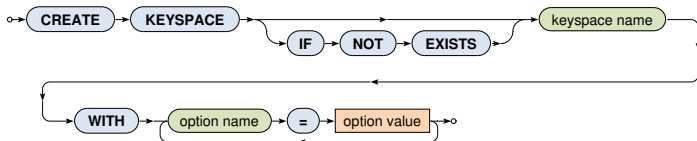
## **CQL = Cassandra Query Language**

- Declarative query language
  - Inspired by SQL
- **DDL statements**
  - CREATE KEYSPACE – creates a new keyspace
  - CREATE TABLE – creates a new table
  - ...
- **DML statements**
  - SELECT – selects and projects rows from a single table
  - INSERT – inserts rows into a table
  - UPDATE – updates columns of rows in a table
  - DELETE – removes rows from a table
  - ...



# Keyspaces

## CREATE KEYSPACE



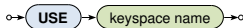
- **Creates a new keyspace**
- **Replication option** is mandatory
  - `SimpleStrategy` (one replication factor)
  - `NetworkTopologyStrategy`  
(individual replication factor for each data center)

```
CREATE KEYSPACE moviedb
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3}
```

# Keyspaces

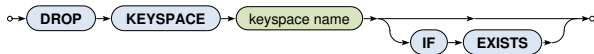
## USE

- Changes the current keyspace



## DROP KEYSPACE

- Removes a keyspace, all its tables, data etc.



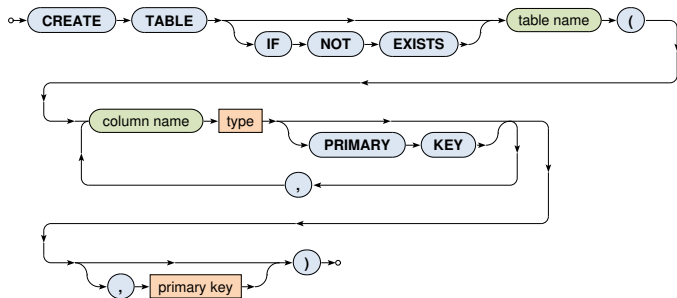
## ALTER KEYSPACE

- Modifies options of an existing keyspace

# Tables

## CREATE TABLE

- **Creates a new table** within the current keyspace
- Each table must have exactly one **primary key** specified



# Tables

## Examples – tables for **actors** and **movies**

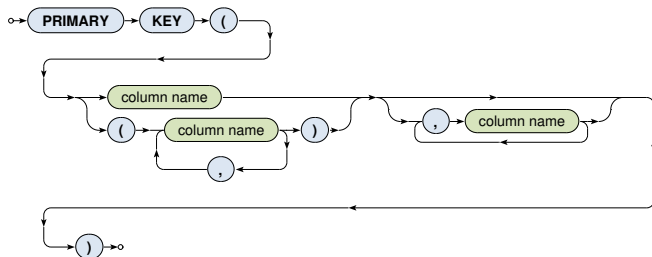
```
CREATE TABLE actors (  
  id TEXT PRIMARY KEY,  
  name TUPLE<TEXT, TEXT>,  
  year SMALLINT,  
  movies SET<TEXT>  
)
```

```
CREATE TABLE movies (  
  id TEXT,  
  title TEXT,  
  director TUPLE<TEXT, TEXT>,  
  year SMALLINT,  
  actors MAP<TEXT, TEXT>,  
  genres LIST<TEXT>,  
  countries SET<TEXT>,  
  PRIMARY KEY (id)  
)
```

# Tables

**Primary key** has two parts:

- Compulsory **partition key**
  - Single column or multiple columns
  - Describes how table rows are distributed among partitions
- Optional **clustering columns**
  - Defines the clustering order, i.e. how table rows are locally stored within a partition



# Tables

## DROP TABLE

- Removes a table together with all data it contains



## TRUNCATE TABLE

- Preserves a table but removes all data it contains



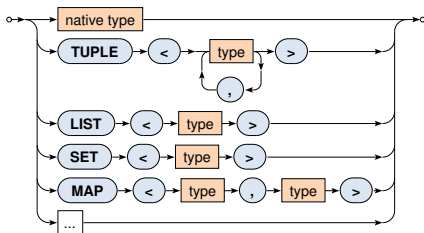
## ALTER TABLE

- Allows to alter, add or drop table columns

# Types

## Types of columns

- Native types
- **Tuples**
- Collection types: **lists**, **sets**, and **maps**
- **User-defined types**



# Types

## Native types

- `tinyint`, `smallint`, `int`, `bigint`
  - Signed integers (1B, 2B, 4B, 8B)
- `varint`
  - Arbitrary-precision integer
- `decimal`
  - Variable-precision decimal
- `float`, `double`
  - Floating point numbers (4B, 8B)
- `boolean`
  - Boolean values `true` and `false`



# Types

## Native types

- `text`, `varchar`
  - UTF8 encoded string
  - Enclosed in single quotes (not double quotes)
    - Escaping sequence: `'`
- `ascii`
  - ASCII encoded string
- `date`, `time`, `timestamp`
  - Dates, times and timestamps
  - E.g. `'2016-12-05'`, `'2016-12-05 09:15:00'`, `1480929300`

# Types

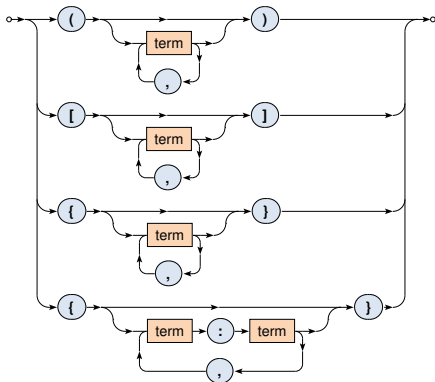
## Native types

- `counter` – 8B signed integer
  - Only 2 operations supported: incrementing and decrementing
    - I.e. value of a counter cannot be set to a particular number
  - Restrictions in usage
    - Counters cannot be a part of a primary key
    - Either all table columns (outside the primary key) are counters, or none of them
    - TTL is not supported
    - ...
- `blob` – arbitrary bytes
- `inet` – IP address (both IPv4 and IPv6)
- ...

# Literals

## Tuple and collection literals

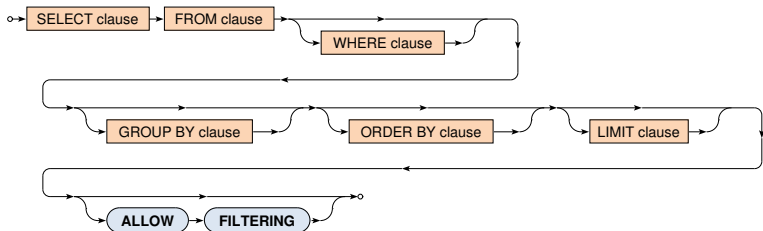
- Literals for **tuples**, **lists**, **sets**, and **maps**



# Selection

## SELECT statement

- **Selects matching rows** from a **single table**



# Selection

## Clauses of SELECT statements

- SELECT – columns or values to appear in the result
- FROM – single table to be queried
- WHERE – filtering conditions to be applied on table rows
- GROUP BY – columns used for grouping of rows
- ORDER BY – criteria defining the order of rows in the result
- LIMIT – number of rows to be included in the result

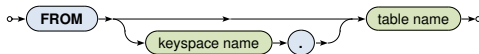
## Example

```
SELECT id, title, actors
FROM movies
WHERE year = 2000 AND genres CONTAINS 'comedy'
```

# Selection

## FROM clause

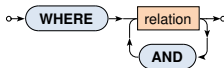
- Defines a **single table to be queried**
  - From the current / specified keyspace
- I.e. joining of multiple tables is not possible



# Selection

## WHERE clause

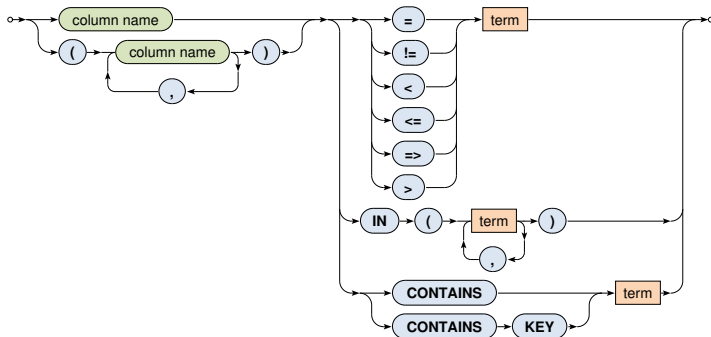
- **One or more relations a row must satisfy** in order to be included in the query result



- Only simple conditions can be expressed and **not all relations are allowed**, e.g.:
  - only primary key columns can be involved unless secondary index structures exist
  - non-equal relations on partition keys are not supported
  - ...

# Selection

## WHERE clause: relations





# Selection

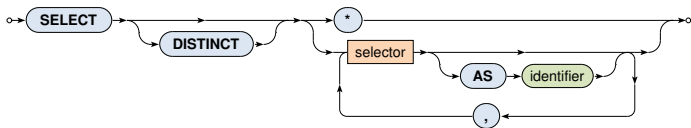
## WHERE clause: relations

- **Comparisons**
  - =, !=, <, <=, =>, >
- **IN**
  - Returns true if the actual value is one of the enumerated
- **CONTAINS**
  - May only be used on collections (lists, sets, and maps)
  - Returns true if a collection contains a given element
- **CONTAINS KEY**
  - May only be used on maps
  - Returns true if a map contains a given key

# Selection

## SELECT clause

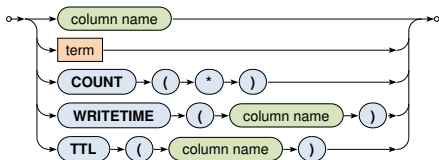
- Defines **columns or values to be included in the result**
  - \* = all the table columns
  - Aliases can be defined using AS



- **DISTINCT** – duplicate rows are removed

# Selection

## SELECT clause: **selectors**

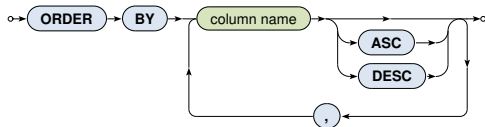


- **COUNT(\*)**
  - Number of all the rows in a group (see aggregation)
- **WRITETIME** and **TTL**
  - Selects timestamp / remaining time-to-live of a given column
  - Cannot be used on collections and their elements
  - Cannot be used in other clauses (e.g. WHERE)

# Selection

## ORDER BY clause

- Defines the **order of rows returned in the query result**
- Only orderings induced by clustering columns are allowed!



## LIMIT clause

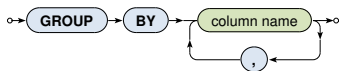
- **Limits the number of rows** returned in the query result



# Selection

## GROUP BY clause

- **Groups rows of a table** according to certain columns
- Only groupings induced by primary key columns are allowed!



- **When a non-grouping column is selected** without an aggregate function, the first value encounter is always returned

# Selection

## **GROUP BY** clause: **aggregates**

- Native aggregates
  - **COUNT**(column)
    - Number of all the values in a given column
    - `null` values are ignored
  - **MIN**(column), **MAX**(column)
    - Minimal / maximal value in a given column
  - **SUM**(column)
    - Sum of all the values of a given column
  - **AVG**(column)
    - Average of all the values of a given column
- User-defined aggregates

# Selection

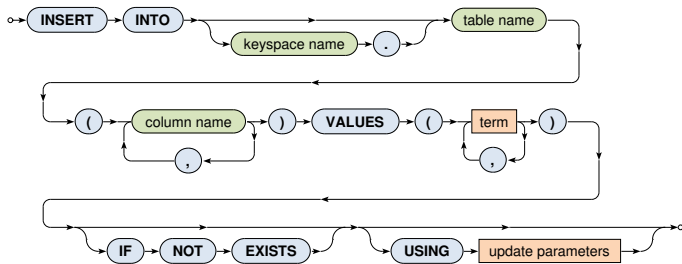
## ALLOW FILTERING modifier

- By default, **only non-filtering queries are allowed**
  - I.e. queries where **the number of rows read  $\sim$  the number of rows returned**
  - Such queries have predictable performance
    - They will execute in a time that is proportional to the amount of data returned
- ALLOW FILTERING **enables (some) filtering queries**

# Insertions

## INSERT statement

- **Inserts a new row** into a given table
  - When a row with a given primary key already exists, it is updated
- At least primary key columns must be specified
  - ... and they have to always be explicitly enumerated





# Insertions

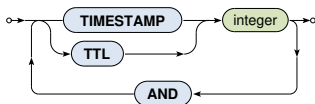
## Example

```
INSERT INTO movies (id, title, director, year, actors, genres)
VALUES (
  'stesti',
  'Štěstí',
  ('Bohdan', 'Sláma'),
  2005,
  { 'vilhelмова': 'Monika', 'liska': 'Toník' },
  [ 'comedy', 'drama' ]
)
USING TTL 86400
```

# Insertions and Updates

## Update parameters

- **TTL**: time-to-live
  - 0 or null or simply missing for persistent values
- **TIMESTAMP**: writetime

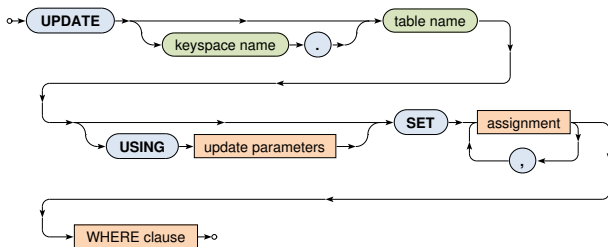


- Only newly inserted / updated values are really affected

# Updates

## UPDATE statement

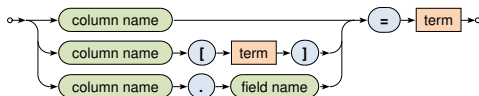
- **Updates existing rows** within a given table
  - When a row with a given primary key does not yet exist, it is inserted
- All primary key columns must be specified in the WHERE clause



# Updates

## UPDATE statement: **assignments**

- Describe modifications to be applied
- Allowed assignments:
  - Value of a whole column is replaced
  - Value of a list or map element is replaced
  - Value of an UDT field is replaced



# Updates

## Examples

```
UPDATE movies
SET
  year = 2006,
  director = ('Jan', 'Svěrák'),
  actors = { 'machacek': 'Robert Landa', 'sverak': 'Josef Tkaloun' },
  genres = [ 'comedy' ],
  countries = { 'CZ' }
WHERE id = 'vratnelahve'
```

```
UPDATE movies
SET
  actors = actors + { 'vilhelmova': 'Helenka' },
  genres = [ 'drama' ] + genres,
  countries = countries + { 'SK' }
WHERE id = 'vratnelahve'
```

# Updates

## Examples

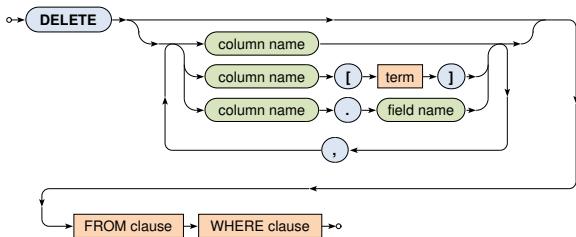
```
UPDATE movies
SET
  actors = actors - { 'vilhelmova', 'landovsky' },
  genres = genres - [ 'drama', 'sci-fi' ],
  countries = countries - { 'SK' }
WHERE id = 'vratnelahve'
```

```
UPDATE movies
SET
  actors['vilhelmova'] = 'Helenka',
  genres[1] = 'comedy'
WHERE id = 'vratnelahve'
```

# Deletions

## DELETE statement

- Removes existing rows / columns / collection elements from a given table



# Lecture Conclusion

Cassandra

- **Column-family store**

Cassandra query language

- DDL statements
- DML statements
  - **SELECT, INSERT, UPDATE, DELETE**