

B4M36DS2, BE4M36DS2: Database Systems 2

<http://www.ksi.mff.cuni.cz/~svoboda/courses/201-B4M36DS2/>

Lecture 9

Wide Column Stores: Cassandra

Martin Svoboda

martin.svoboda@fel.cvut.cz

23. 11. 2020

Charles University, Faculty of Mathematics and Physics

Czech Technical University in Prague, Faculty of Electrical Engineering

Lecture Outline

Wide column stores

- Introduction

Apache Cassandra

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

Wide Column Stores

Data model

- Column family
 - 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

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**
 - Rows do not need to have exactly the same columns
 - Table schema must be specified, yet can be modified later on
- Row
 - **Collection of columns**
 - Each row is **uniquely identified** by a compulsory **primary key**
- Column
 - **Name-value pair** + additional data

Data Model

Column values

- Empty value
 - `null`
- Atomic values
 - **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 also permitted, however, currently only in a **frozen mode**

Data Model

Collections

- **List = ordered collection of values**
 - This order is based on positions
 - Values do not need to be unique
- **Set = collection of unique values**
 - Values are internally ordered
- **Map = collection of key-value pairs**
 - Keys must be unique
 - Pairs are internally ordered based on keys

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' | <table border="1"><tr><td>title</td><td>year</td><td>actors</td><td>genres</td></tr><tr><td>'Samotáři'</td><td>2000</td><td>null</td><td>['comedy', 'drama']</td></tr></table> | title | year | actors | genres | 'Samotáři' | 2000 | null | ['comedy', 'drama'] | | | | |
| title | year | actors | genres | | | | | | | | | | |
| 'Samotáři' | 2000 | null | ['comedy', 'drama'] | | | | | | | | | | |
| 'medvidek' | <table border="1"><tr><td>title</td><td>director</td><td>year</td></tr><tr><td>'Medvídek'</td><td>('Jan', 'Hřebejk')</td><td>2007</td></tr><tr><td>properties</td><td colspan="2">actors</td></tr><tr><td>{ length: 100 }</td><td colspan="2">{ 'trojan': 'Ivan', 'machacek': 'Jirka' }</td></tr></table> | title | director | year | 'Medvídek' | ('Jan', 'Hřebejk') | 2007 | properties | actors | | { length: 100 } | { 'trojan': 'Ivan', 'machacek': 'Jirka' } | |
| title | director | year | | | | | | | | | | | |
| 'Medvídek' | ('Jan', 'Hřebejk') | 2007 | | | | | | | | | | | |
| properties | actors | | | | | | | | | | | | |
| { length: 100 } | { 'trojan': 'Ivan', 'machacek': 'Jirka' } | | | | | | | | | | | | |
| 'vratnelahve' | <table border="1"><tr><td>title</td><td>year</td></tr><tr><td>'Vratné lahve'</td><td>2006</td></tr></table> | title | year | 'Vratné lahve' | 2006 | | | | | | | | |
| title | year | | | | | | | | | | | | |
| 'Vratné lahve' | 2006 | | | | | | | | | | | | |
| 'zelary' | <table border="1"><tr><td>title</td><td>year</td><td>actors</td><td>genres</td></tr><tr><td>'Želary'</td><td>2003</td><td>{ }</td><td>['romance', 'drama']</td></tr></table> | title | year | actors | genres | 'Želary' | 2003 | { } | ['romance', 'drama'] | | | | |
| title | year | actors | genres | | | | | | | | | | |
| 'Želary' | 2003 | { } | ['romance', 'drama'] | | | | | | | | | | |

Data Model

Additional data associated with...

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

- **Time-to-live (TTL)**
 - After a certain period of time (number of seconds) a given column / element is automatically deleted
- **Timestamp (writetime)**
 - Timestamp of the last 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

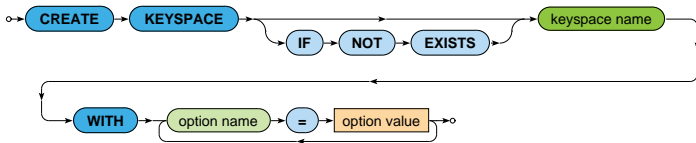
CQL = Cassandra Query Language

- 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
 - ...

DDL Statements

Keyspaces

CREATE KEYSPACE



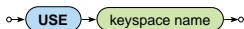
- **Creates a new keyspace**
- **Replication option is mandatory**
 - `SimpleStrategy` (only one replication factor)
 - `NetworkTopologyStrategy` (individual replication factor for each data center)

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

Keyspaces

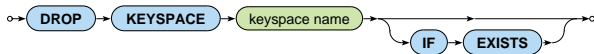
USE

- Changes the current keyspace



DROP KEYSPACE

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



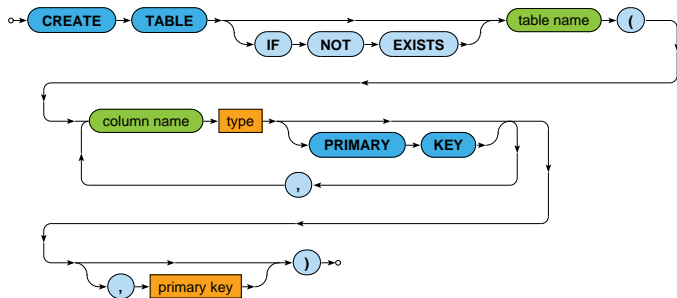
ALTER KEYSPACE

- 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



- None of the columns is compulsory (except the primary key)

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>,  
  properties details,  
  PRIMARY KEY (id)  
)
```

Primary Keys

Primary keys have two parts

- Compulsory **partition key**
 - At least one column
 - Defines how individual rows are distributed between shards
- Optional **clustering columns**
 - Defines the order in which individual rows are locally stored by each shard

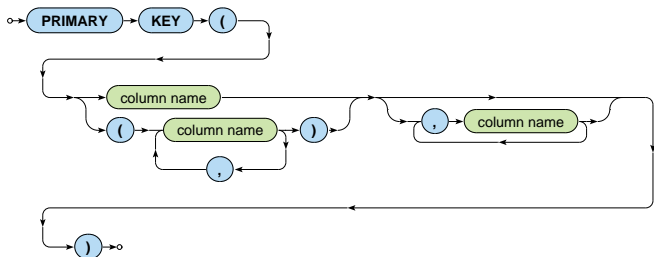
Column-level primary key definition

- A given column (the only one) becomes the partition key
- There are no clustering columns

Primary Keys

Table-level primary key definition

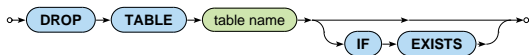
- The first column / all columns in the embedded parentheses become the partition key
- All the remaining ones (if any) form the clustering columns



Tables

DROP TABLE

- Removes a table together with all data it contains



TRUNCATE TABLE

- Preserves a table but removes all data it contains



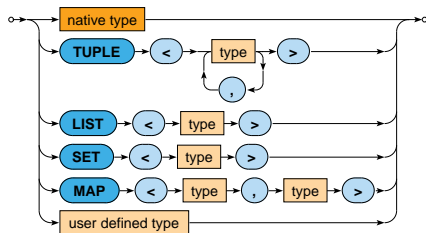
ALTER TABLE

- Allows to alter, add or drop table columns

Data Types

Types of columns

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



Native Data 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`

Native Data 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`

Native Data Types

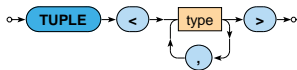
Native types

- `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)
- ...

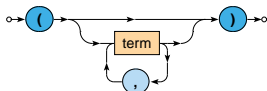
Tuple Data Types

Tuples

- Declaration



- Literals



- E.g. ('Jiří', 'Macháček')

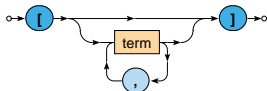
Collection Data Types

Lists

- Declaration



- Literals



- E.g. ['comedy', 'drama']

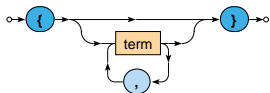
Collection Data Types

Sets

- Declaration



- Literals

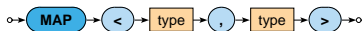


- E.g. { 'medvidek', 'vratnelahve', 'samotari' }

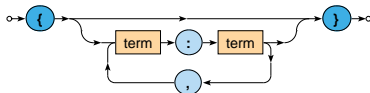
Collection Data Types

Maps

- Declaration



- Literals

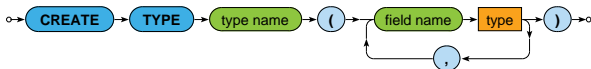


- E.g. { 'machacek': 'Robert Landa' }

User-Defined Data Types

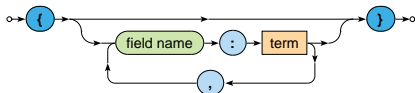
User-defined types (UDT)

- Definition



- E.g. `CREATE TYPE details (length SMALLINT, annotation TEXT)`

- Literals



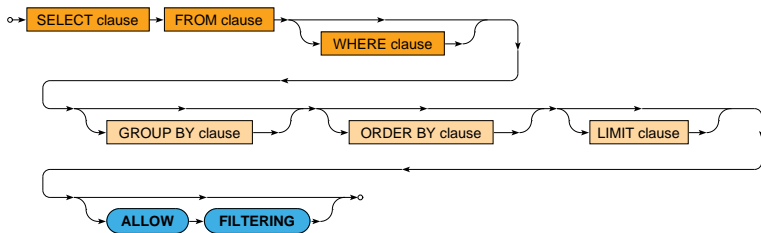
- E.g. `{ length: 100 }`

DML Statements

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 to be 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

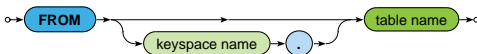
Example

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

Selection

FROM clause

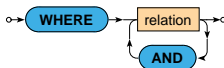
- Defines a **single table to be queried**
 - From the current / selected keyspace
- I.e. joining of multiple tables is not possible



Selection

WHERE clause

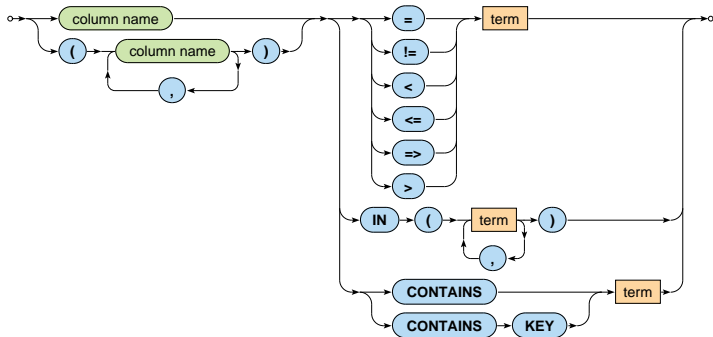
- **One or more relations a row must satisfy** in order to be included in the query result



- 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

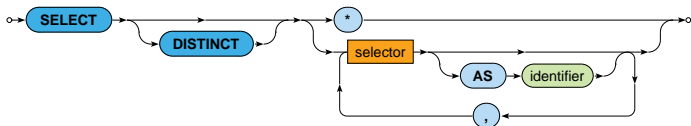
WHERE clause: relations

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

Selection

SELECT clause

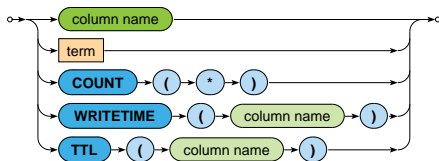
- Defines **columns or values to be included in the result**
 - * = all the table columns
 - Aliases can be defined using AS



- **DISTINCT** – duplicate rows are removed

Selection

SELECT clause: selectors

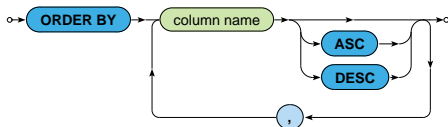


- **COUNT(*)**
 - Number of all the rows in a group (see aggregation)
- **WRITETIME** and **TTL**
 - Selects modification timestamp / remaining time-to-live of a given column
 - Cannot be used on collections and their elements
 - Cannot be used in other clauses (e.g. WHERE)

Selection

ORDER BY clause

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



LIMIT clause

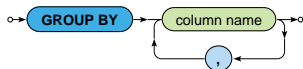
- **Limits the number of rows** returned in the query result



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 would be accessed directly** in the SELECT clause (i.e. without being wrapped by an aggregate function), the first value encountered will always be 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 in a given column
 - **AVG**(column)
 - Average of all the values in a given column
- User-defined aggregates

Selection

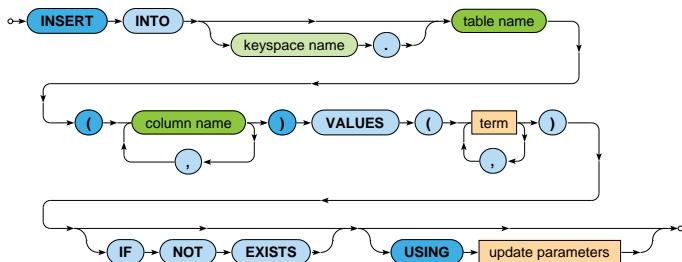
ALLOW FILTERING modifier

- 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
- Values of at least primary key columns must be set
- Names of columns must always be explicitly enumerated



Insertions

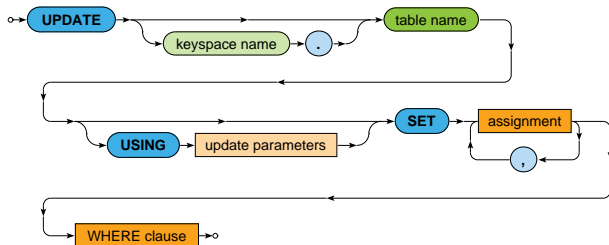
Example

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

Updates

UPDATE statement

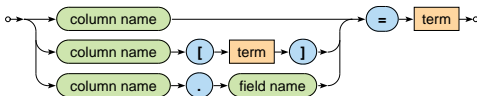
- **Updates existing rows** within a given table
 - When a row with a given primary key does not yet exist, it is inserted
- At least all primary key columns must be specified in the WHERE clause



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
 - Items of lists are numbered starting with 0
 - Value of a user-defined type 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['vilhelmova'] = 'Helenka',
  genres[1] = 'comedy',
  properties.length = 99
WHERE id = 'vratnelahve'
```


Updates

Examples: modification of collection elements

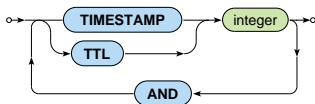
```
UPDATE movies
SET
  actors = actors + { 'vilhelmova': 'Helenka' },
  genres = [ 'drama' ] + genres,
  countries = countries + { 'SK' }
WHERE id = 'vratnelahve'
```

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

Insertions and Updates

Update parameters

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

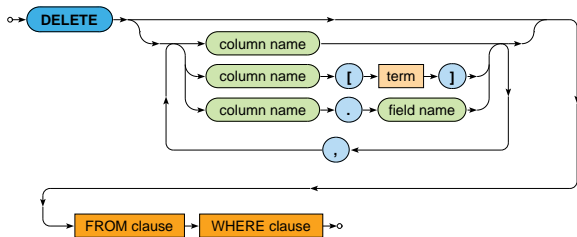


- Only newly inserted / updated values are really affected

Deletions

DELETE statement

- **Removes the matching rows /**
Preserves these rows but **removes the selected columns /**
Preserves these columns but **removes elements of collections**
or **fields of UDT values**



Lecture Conclusion

Cassandra

- **Wide column store**

Cassandra query language

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