University “St. Kliment Ohridski” - Bitola, Macedonia
Faculty of Information and Communication Technologies – Bitola
Department of Software Engineering and Information Systems

3. Column-Oriented Databases
Cassandra & Cassandra Query Language CQL

Prof. Blagoj Ristevski, PhD

JKU - Johannes Kepler Universität Linz, Austria
Institut für Systemsoftware, November 2024

Column-oriented DBs

- Column-oriented DBs
  - Data as tables :

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	60000
002	12	Jones	Mary	80000
003	11	Johnson	Cathy	94000
004	22	Jones	Bob	55000

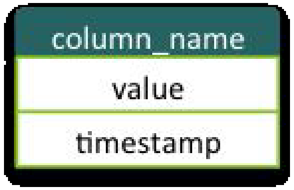
- They are stored as columns:

10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
60000:001,80000:002,94000:003,55000:004

- examples: SAP HANA, Apache Cassandra, Apache Kudu, C-Store, ...
- Rows that have many columns associated with a row key
  - can be written as JSON

Data Model

- **Column =** the basic data item
  - a 3-tuple consisting of
    - column **name**
- **value**
- **timestamp**



- Can be modelled as follows

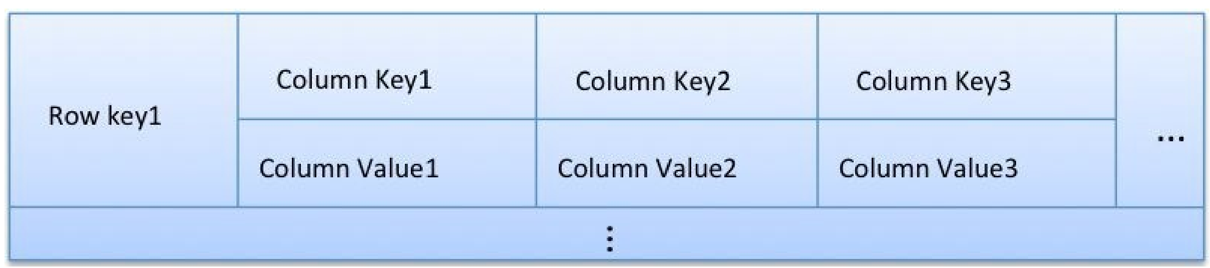
```
{ name: "firstName",  value: "Martin",  timestamp:
    12345667890 }
```

- In the following, we will ignore the timestamp

Data Model

- **Row:** a collection of columns attached to row key
  - Columns can be added to any row at any time
    - without having to add it to other rows

```
// row
"martin-fowler" : {  firstName: "Martin",  lastName:
    "Fowler",  location: "Boston"
}
```



Data Model

- Column Family
- **CF =** Set of columns containing “related” data

user_id (*row key*)	*column key*	*column key*	...
	column value	*column value*	...
1	login	first_name	...
	honza	Jan	...
4	login	age	...
	david	35	...
5	first_name	last_name	...
	Irena	Holubová	...
...			

Data Model

- ❑ Column family - example as JSON

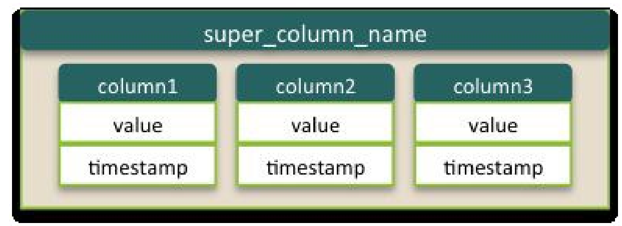
```
{ // row (columns from a CF)
  "pramod-sadalage" : {
firstName: "Pramod",
  lastName: "Sadalage",
  lastVisit: "2012/12/12"
}
```

```
// row (cols from the same CF)
  "martin-fowler" : {
firstName: "Martin",
  lastName: "Fowler",
  location: "Boston",  activ:
  "true" }
}
```

Data Model

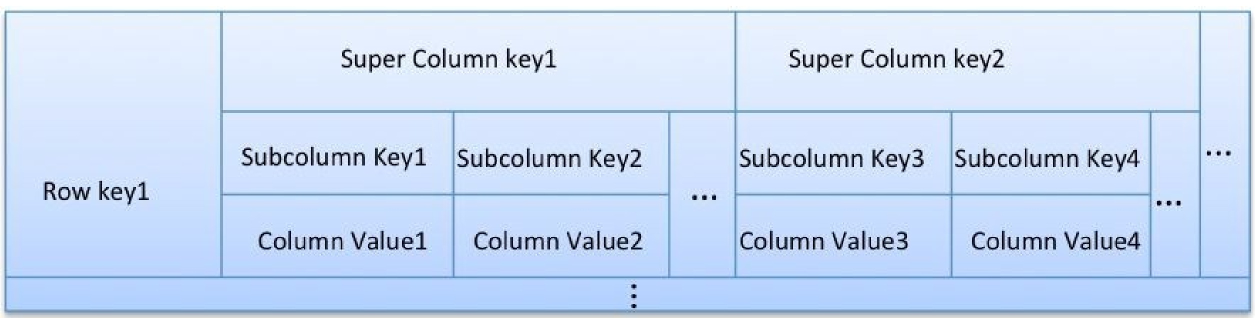
❑ Super column

- A column whose value is composed of a map of columns
- Used in some column-family stores (Cassandra 1.0)



❑ Super column family

- A column family consisting of super columns



Super Column Family

Addresses

user_id (*row key*)	*super column key*			*super column key*			*...*
	subcolumn key	*subcolumn key*	...	*subcolumn key*	*subcolumn key*	...	...
	subcolumn value	*subcolumn value*	...	*subcolumn value*	*subcolumn value*	...	
1	home_address			work_address			
	city	street	...	city	street	...	
	Brno	Krásná 5	...	Praha	Pracovní 13	...	
4	home_address			temporary_address			
	city	street	...	city	PSČ		
	Plzeň	sady Pětatřicátníků 35	...	Praha	111 00		
...							

Super Column Family

Generic

```
{ // row  "Cath": {
"username": { "firstname": "Cath", "lastname": "Yoon" },
"address": { "city": "Seoul", "postcode": "1234" }
}
// row  "Terry": {
"username": { "firstname": "Terry", "lastname": "Cho" },
"account": { "bank": "Hana", "accounted": 1234 },
"preferences": { "color": "blue", "style": "simple" }
}
}
```

Data Model

1. Each column family = a relational table
   - with (a lot of) **null** values

	Cassandra Model
	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Data Model

Column family = a map of maps (nested map)

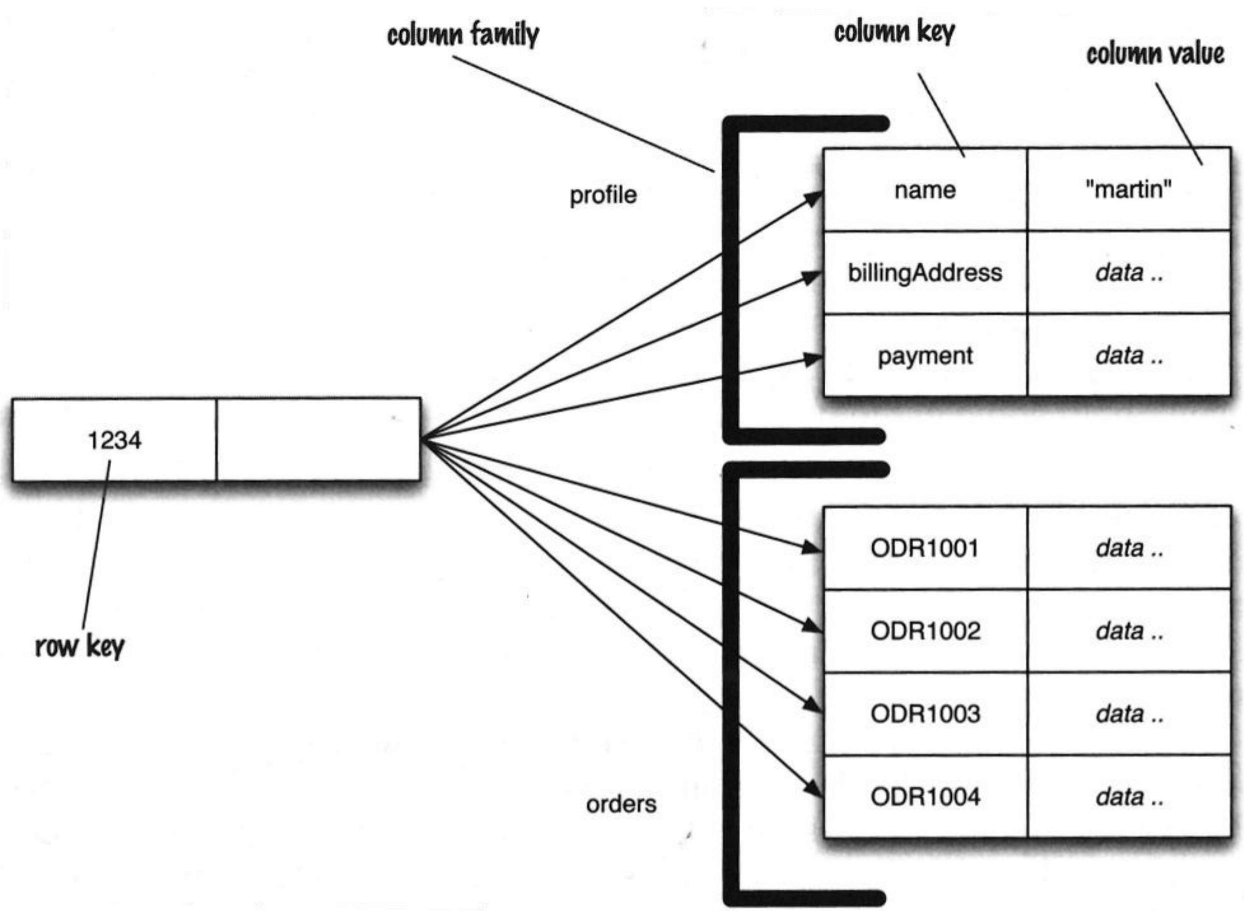
```
Map<RowKey, Map<ColumnKey, ColumnValue>>
```

- Super column family:

```
Map<RowKey, Map<SuperColumnKey,
                Map<ColumnKey, ColumnValue>>>
```

- The column-family data model can be viewed as *JSON-like documents with restrictions on the format*

Example



column family
column key
column value
profile
name
"martin"
billingAddress
data ..
payment
data ..
1234
row key
ODR1001
data ..
ODR1002
data ..
ODR1003
data ..
ODR1004
data ..
orders

Cassandra

- Developed at Facebook
  - now under Apache Software License 2.0
- Initial release: 2008 (stable release 4.1 in 2024)
- Written in Java
- OS: cross-platform
- Operations:
  - CQL (Cassandra Query Language)
  - MapReduce support (can cooperate with Hadoop)
- Professional support by DataStax
  - http://www.datastax.com/

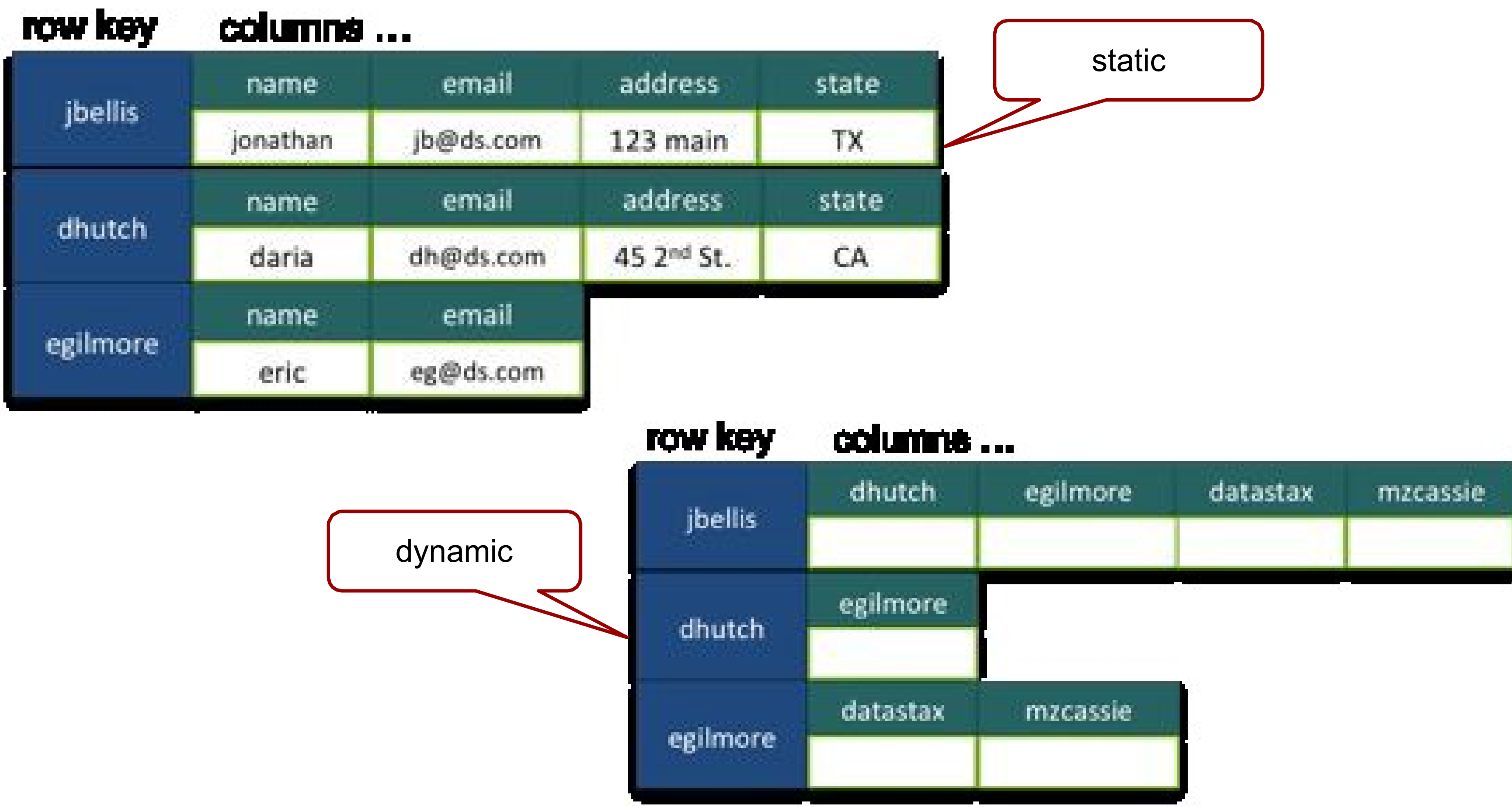
Cassandra 1.0: Data Model

- Column families, super column families
    - Can define metadata about columns
    - Now denoted as: Thrift API

- **Static** – similar to a relational database table
    - Rows have the same set of columns
        - Not required to have all of the columns set

- **Dynamic** – takes advantage of Cassandra's ability to use arbitrary column names

Cassandra 1.0: Data Model



Cassandra 1.0: Column Families

- A key *must* be specified
- Data types for columns *can* be specified
- Options *can* be specified

```
CREATE COLUMNFAMILY Fish (key blob PRIMARY KEY);
CREATE COLUMNFAMILY FastFoodEatings (user text PRIMARY KEY)
   WITH comparator=timestamp AND default_validation=int;
CREATE COLUMNFAMILY MonkeyTypes (  key uuid PRIMARY KEY,
   species text, alias text,  population varint
)  WITH comment='Important biological records'
AND read_repair_chance = 1.0;
```

Cassandra 1.0: Column Families

- **Comparator** = data type for a column name
- **Validator** = data type of a column value
  - or content of a row key

- Data types do not need to be defined
  - Default: BytesType, i.e. arbitrary hexadecimal bytes

- Basic operations: GET, SET, DEL

Data Manipulation

```
create column family users
with key_validation_class = Int32Type
and comparator = UTF8Type
and default_validation_class = UTF8Type;

// set column values in row with key 7  set
users[7]['login'] = utf8('honza');  set
users[7]['name'] = utf8('Jan Novák');
set users[7]['email'] = utf8('jan@novak.name');

set users[13]['login'] = utf8('fantomas');
set users[13]['name'] = utf8('incognito');
```

Data Manipulation

```
get users[7]['login'];
=> (name=login, value=honza, timestamp=1429268223462000)

get users[13];
=> (name=login, value=fantomas, timestamp=1429268224554000)
=> (name=name, value=incognito, timestamp=1429268224555000)

list users;
RowKey: 7
=> (name=email, value=jan@novak.name, timestamp=14292682...)
=> (name=login, value=honza, timestamp=1429268223462000)
=> (name=name, value=Jan Novák, timestamp=1429268223471000)
-------------------
RowKey: 13
=> (name=login, value=fantomas, timestamp=1429268224554000)
```

Cassandra: Sparse Tables

- CQL: Cassandra Query Language
  - SQL-like commands
    - CREATE, ALTER, UPDATE, DROP, DELETE, TRUNCATE, INSERT, …
  - Simpler than SQL

- Since CQL 3 (Cassandra 1.2)
  - Column -> cell
  - Column family -> table
- Dynamic columns (wide rows) still supported

Working with Tables

```
CREATE TABLE users (
user_id int PRIMARY KEY,  login text,
name text,  email text );

INSERT INTO users (user_id, login, name)
VALUES (3, 'honza', ‘John');



SELECT * FROM users;
 user_id | email | login | name
---------+-------+-------+----------
       3 | null  | honza | John
```

Tables: Dynamic Columns

- Values can use "collection" types:
  - **set** – unordered unique values
  - **list** – ordered list of elements
  - **map** – name + value pairs
    - a way to realize super-columns
- Realization of the original idea of free columns
  - Internally, all values in collections as individual columns
  - Cassandra can handle "unlimited" number of columns well

Tables: Dynamic Columns

```
CREATE TABLE users (
login text PRIMARY KEY,  name text,
emails set<text>, // column of type “set”  profile
map<text, text> // column of type “map”
)

INSERT INTO users (login, name, emails, profile)
VALUES ( 'honza', 'Jan Novák', { 'honza@novak.cz' },
{ 'colorschema': 'green', 'design': 'simple' }
);

UPDATE users
SET emails = emails + { 'jn@firma.cz' }
WHERE login = 'honza';
```

Dynamic Columns: Another Way

- Compound primary key

```
CREATE TABLE mytable (
    row_id int, column_name text, column_value text,
    PRIMARY KEY (row_id, column_name)
);

INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'login', 'honza');
INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'name', 'Jan Novák');
INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'email', 'honza@novak.cz');
```

Cassandra Query Language (CQL)

- The syntax of CQL is similar to SQL
  - But search just in one table (no joins)

```
SELECT <selectExpr>
FROM [<keyspace>.]<table>  [WHERE
   <clause>]
[ORDER BY <clustering_colname>
   [DESC]]
[LIMIT m];
```

```
SELECT column_name, column_value
FROM mytable
WHERE row_id=3
ORDER BY column_name;
```

```
CREATE TABLE mytable (  row_id int,
     column_name text,  column_value
     text,
PRIMARY KEY (row_id, column_name)
);
```

CQL: Limitations on “Where” Part

```
CREATE TABLE mytable (
  row_id int,  column_name
  text,  column_value text,
  PRIMARY KEY
(row_id, column_name )
);
```

- The search condition can be:
  - on columns in the partition key
    - And only using operators = and IN

```
... WHERE row_id IN (3, 4, 5)
```

    - Therefore, the query hits only one or several physical nodes (not all)

  - on columns from the clustering key
    - Especially, if there is also condition on the partitioning key

```
... WHERE row_id=3 AND column_name='login'
```

    - If it is not, the system must filter all entries

```
SELECT * FROM mytable
WHERE column_name IN ('login', 'name') ALLOW FILTERING;
```

Cassandra - Summary

- Column-family stores
  - are worth only for large data and large query throughput
  - two ways to see the data model:
    - large sparse tables or multidimensional (nested) maps
  - data distribution is via row key
    - analogue of document ID or key in a document or key-value stores
  - efficient disk + memory local data storage

- Cassandra
  - CQL: structured after SQL, easy transition from RDBMS

Thank you for your attention!

Do you have any questions?