International Journal of Computer Engineering and Data Science (ISSN: **2737-8543**)
Volume 1– Issue 1, 20/07/2021

Methodology for modeling a column-oriented database with Cassandra

Ahmed DOURHRI[1], Mohamed HANINE[2] and Hassan OUAHMANE[3]
The Information Technology Laboratory
National School of Applied Sciences, Chouaib Doukkali University, El Jadida, Morocco
[1]dourhriahmed@gmail.com , [2]m.hanine.ensaj@gmail.com , [3]hassan.ouahmane@yahoo.fr

Article history

Received May 19, 2021
Revised June 09, 2021
Accepted July 03, 2021
Published July 20, 2021

ABSTRACT

Cassandra data model is based on a dynamic schema, with a column-oriented data model. This means that, unlike a relational database, it is not necessary to model all the columns since a row potentially does not have the same set of columns. Columns and their metadata can be added by the application when needed. On the other hand, although the tables of a Cassandra database can be flexible, in practice, it is advisable to associate some kind of schema. This paper aims to explain a methodology for developing the physical model of a column-oriented database from the data circulating in an information system, a model also driven by the queries.

Keywords: Data modeling, NoSQL, column-oriented model, Cassandra, query driven model.

I. INTRODUCTION

With the increase in Internet bandwidth, as well as the decrease in costs of computer equipment, new possibilities have emerged in the field of distributed computing. The passage to the 21st century by the WEB 2.0 revolution has seen the data volume of some companies increase the process. These data, mainly, from social networks, database medical, economic indicators, etc. Continuous computerization of processing of any kind has resulted in an exponential increase in this volume of data, which is now counted in petabytes, the Anglo-Saxons have called it the Big Data. The management of these volumes of data has therefore become a problem that relational databases were no longer able to handle. NoSQL brings together therefore many databases that are no longer based on the logic of relational representation. However, it is not easy to explicitly define this that is a NoSql database as no standard has yet been established.

The fundamental need that NoSQL meets is performance to solve the problems related to "Big data". It is important to know that SQL is not a relational model, but a data manipulation language designed around the model relational. NoSQL databases do not aim to move away from this language but rather from the relational model. A need has emerged to design a robust schema for a database considering, in addition, data useful to the information system, any requests that will be expressed subsequently: this is the subject of this research paper. To illustrate this modeling, we have chosen CASSANDRA as NoSQL oriented-column DBMS (Database Management System). The process of this modeling consists of 3 steps: identify useful data in the form of a conceptual data model, capture the different queries that will be expressed, and create the logical and physical model of the database. The main reasons that led us to choose Cassandra as a target database are its various advantages:

- Very fast to handle a large volume of data.
- Flexible data schemas
- Possibility to put in cluster several Cassandra servers.
- Data replication.

Before presenting the steps of such modeling, we will focus on relational databases using the famous SQL language for querying and NoSQL databases. Each NoSQL system has its language.

Relational database [1]: a relational database is a database that relies on the relational model based on set theory and having been invented by the mathematician Edgar Codd in 1970. This model consists of dividing the data into tables formed rows and columns possibly interconnected by foreign keys. The systems managing such databases are called relational database management

systems (RDBMS). They use for data manipulating the SQL, the abbreviation of Structured Query Language. The strong point of these systems is the optimization of the memory space necessary for data management (to the detriment of query execution time).

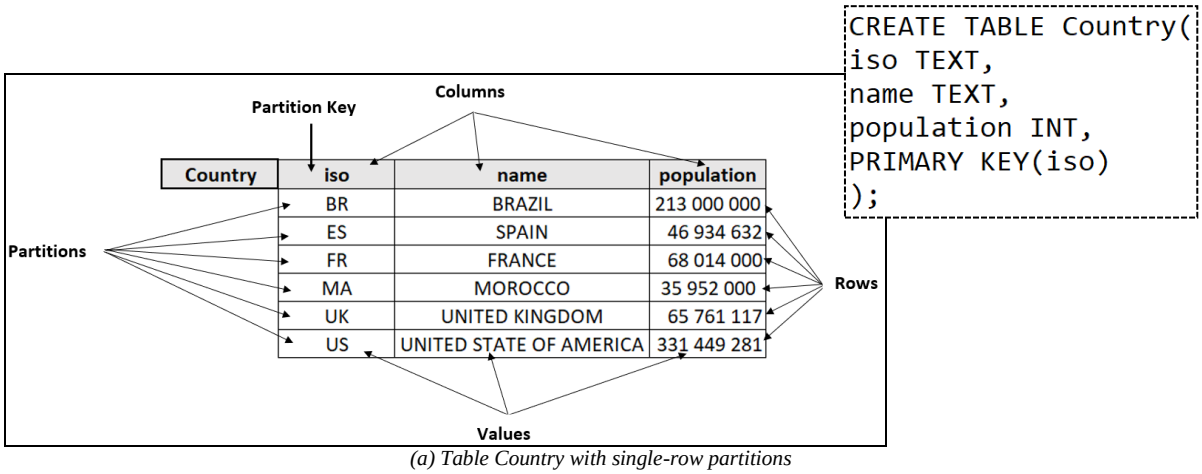
NoSQL database [1]: NoSQL systems come to speed up the querying of databases by offering the possibility of creating flexible schemes and guarantee scalability if necessary. Several models are available for these systems.

- The key-value model: data is stored in the form of a dictionary. This type allows keeping high performance in reading and/or writing. Several products, opting for this model, exist as Voldemort, Redis, Riak…
- Document model [6]: this model extends the key values model in the sense that a database is a set of documents represented in a standard format (JSON: Javascript Object Notation, BSON, etc.). The advantage of this model is the recovery of a set of hierarchically structured information from a single key. Among the well-known implementations of this model, we cite MongoDB and CouchDB.
- Graph model: graph-oriented databases are those that store records in nodes and relationships between records by edges. This type of database is very efficient where there are a huge amount of data strongly connected. Neo4j, HyperGraphDB are examples of graph-oriented DBMS.
- Column-oriented model [7] the notion of a table in this model is different from the notion of a table in a relational database. In this model, a table can be viewed as a set of partitions that contain rows with a similar structure. Each partition in a table has a unique partition key and each row in a partition may optionally have a unique clustering key. Both keys can be simple i.e. one column or composite i.e. multiple columns. The combination of a partition key and a clustering key uniquely identifies a row in a table and is called a primary key. To illustrate some of these notions, **Figure 1** shows tables with CQL (Cassandra Query language) definitions and sample rows. In Figure 1(a), the Country table contains single-row partitions. Its primary key consists of one column iso (iso code of a country) that is also a simple partition key. This table is shown to have six single-row partitions. In Figure 1(b), the EpidemicDataByCountry [5] table contains multi-row partitions. Its primary key consists of partition key (iso) and clustering key observationDate. This table is shown to have six partitions, each one containing multiple rows. For any given partition, its rows are ordered by observationDate in ascending order.

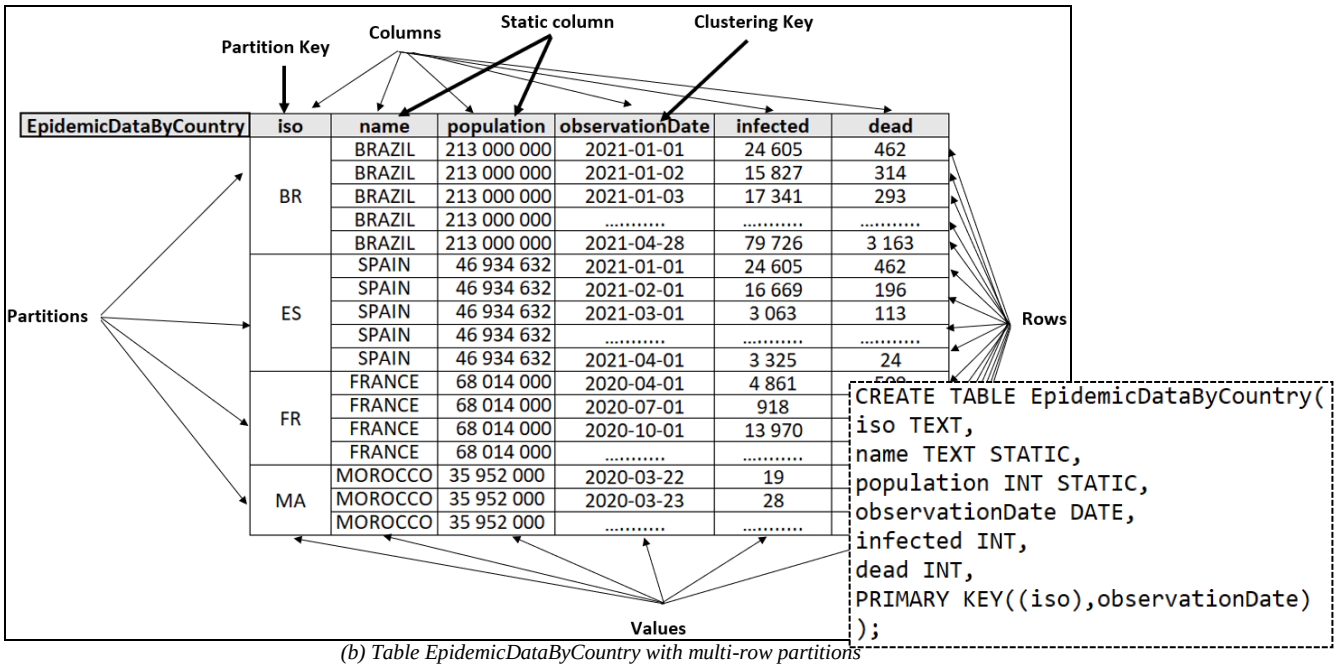
Figure 2 shows the model structure of Cassandra, and **Table 1** shows some SQL concepts and their correspondences in Cassandra.

II. PROPOSED METHODOLOGY

The main aim of this study is the presentation of a methodology for facilitating the modeling of a column-oriented database (case of Cassandra). We propose a query-driven data modeling methodology for Apache Cassandra [2]. A Cassandra solution architect starts data modeling by building a conceptual data model and defining an application workflow to capture all application interactions with a database. The application workflow describes access patterns or queries that a data-driven application needs to run against the database. Then, the architect maps the conceptual data model to a logical data model. The logical data model specifies Cassandra tables that can support application queries, efficiently, according to the application workflow. Finally, additional physical optimizations are applied to produce a physical data model that can be used for Apache Cassandra. The steps of the methodology are visually displayed in Figure 3, which, as highlighted in blue circles, explain how the methodology works. To present each step, we will use an example of a database representing the epidemic data for Coronavirus diseases (COVID-19) in the world by country in Figure 4. The application workflow in Figure 5 models a web application to get data using well-defined queries.



(a) Table Country with single-row partitions



(b) Table EpidemicDataByCountry with multi-row partitions

Figure 1. Sample tables in Cassandra

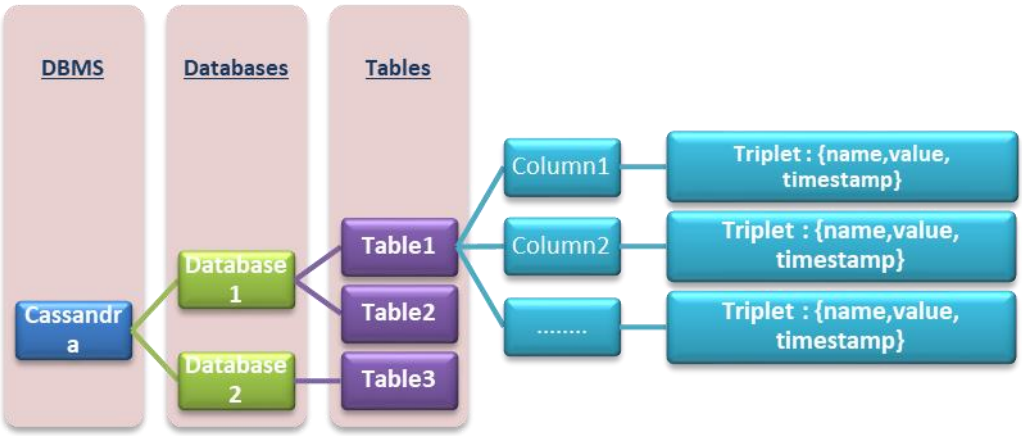


Figure 2. Model structure of Cassandra

TABLE I. Basic SQL vs Cassandra Syntax

SQL terminology [4]	Cassandra Terminology [3]
Database	Database (or keyspace)
Table	Table (or column family)
Row	Row
Column	Column
Primary key	Primary key
join of tables	Nesting with collections (by the use of sets, maps, or lists)
CREATE TABLE Drug(code INT, name TEXT, laboratory TEXT, publicPrice REAL, PRIMARY KEY (code));	CREATE TABLE Drug(code INT, name TEXT, laboratory TEXT, publicPrice REAL, PRIMARY KEY (code));
INSERT INTO Drug (code,name,laboratory,publicPrice) VALUES (1745, 'doliprane 500mg','sanofi', 15.30);	INSERT INTO Drug (code,name,laboratory,publicPrice) VALUES (1745, 'doliprane 500mg','sanofi', 15.30);
CREATE INDEX ON Drug (name);	CREATE INDEX ON Drug (name);
ALTER TABLE Drug ADD dayliDose INT;	ALTER TABLE Drug ADD dayliDose INT;
UPDATE Drug SET publicPrice = publicPrice*1.1 WHERE laboratory=’sanofi’;	UPDATE Drug SET publicPrice = publicPrice*1.1 WHERE laboratory=’sanofi’;
DELETE FROM Drug WHERE code=1439;	DELETE FROM Drug WHERE code=1439;
SELECT * FROM Drug WHERE laboratory = 'COOPER' and publicPrice<90;	SELECT * FROM Drug WHERE laboratory = 'COOPER' and publicPrice<90;
DELETE FROM Drug;	TRUNCATE Drug;

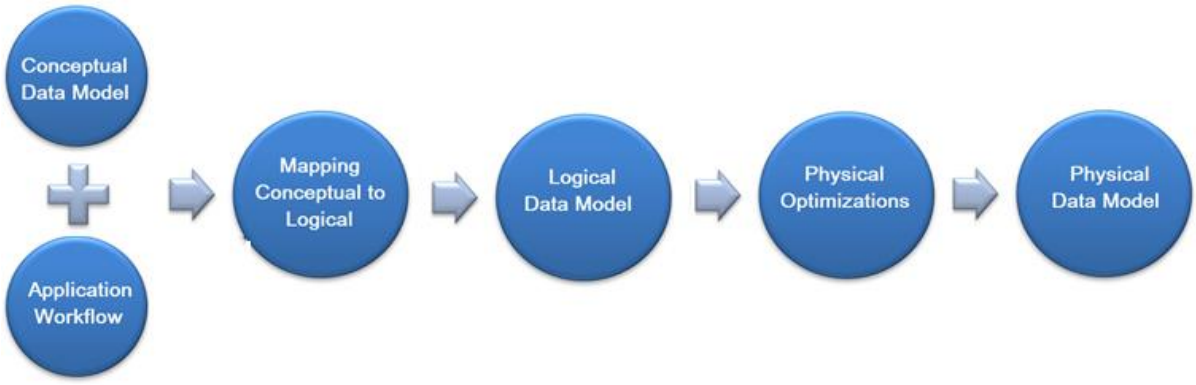


Figure 3. Cassandra Data Modeling

2.1. Conceptual Model

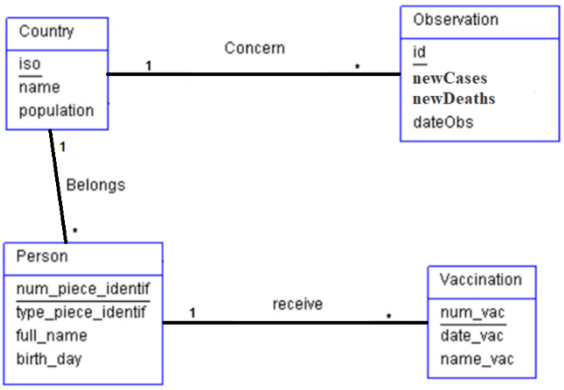


Figure 4. Conceptual Data Model of COVID-19 database

Initially, you will have to provide a detailed data dictionary and then the conceptual data model of the database that we want to create. In this paper, we have used a UML class diagram to represent this conceptual level.

The first key to successful database design is understanding the data, which is captured with a conceptual data model. The importance and effort required for conceptual schema should not be underestimated.

An entity, a relationship, and an attribute types on an ER (Entity Relationship) diagram not only define which data pieces need to be stored in a database but also which data properties, such as entity type and relationship type keys, need to be preserved and relied on to organize data correctly.

2.2. Query model

Like data, queries directly affect table schema design, and if our use case assumptions about the queries change, a database schema will have to change. We define the three broad access paths:

1) partition per query, 2) partition+ per query, and 3) table or table+ per query.

The most efficient option is the “partition per query”, when a query only retrieves one row, a subset of rows, or all rows from a single partition.

Queries over tables are expressed in CQL, which has an SQL-like syntax. Unlike SQL, CQL supports no binary operations, such as joins, and has many rules for query predicates that ensure efficiency and scalability:

- Only primary key columns may be used in a query predicate.
- All partition key columns must be restricted by values (i.e. equality search).
- All, some, or none of the clustering key columns can be used in a query predicate.
- If a clustering key column is used in a query predicate, then all clustering key columns that precede this clustering column in the primary key definition must also be used in the predicate.
- If a clustering key column is restricted by an inequality search in a query predicate, then all clustering key columns that precede this clustering column in the primary key definition must be restricted by values and no other clustering column can be used in the predicate.

Intuitively, a query that restricts all partition key columns by values returns all rows in a partition identified by the specified partition key. For example, the following query over EpidemicDataByCountry table in Figure1(b) returns all statistics about the Brazil country:

SELECT population,observationDate,newCases,newDeaths

FROM *EpidemicDataByCountry*

WHERE iso=‘BR’**:**

A query that restricts all partition key columns and some clustering key columns by values returns a subset of rows from a partition that satisfy such a predicate. Similarly, a query that restricts all partition key columns by values and one clustering key column by range (preceding clustering key columns are restricted by values) returns a subset of rows from a partition that satisfy such a predicate. For example, the following query over the EpidemicDataByCountry table in Figure1(b) returns Brazil statistics for January 2021.

SELECT population,observationDate,newCases,newDeaths

FROM EpidemicDataByCountry

WHERE iso=‘BR’ **AND** observationDate>=‘2021-01-01’

AND observationDate<=‘2021-01-31’;

Query results are always ordered based on the default order specified for clustering key columns when a table is defined (the CLUSTERING ORDER construct), unless a query explicitly reverses the default order (the ORDER BY construct). In the end, CQL supports a number of other features, such as queries that use secondary indexes, IN, and ALLOW FILTERING constructs. Our data modeling methodology does not rely directly on such queries as their performance is unpredictable on large datasets. Figure 5 illustrates a simple example of an application workflow for managing covid-19 epidemic data. The application workflow captures and shows queries that this database will need to support. For readability, in this paper, we use verbal descriptions of access patterns.

The sequence of workflow steps matters because it helps us determine what data is available and required for each query. For example, before we can show statistics about the covid-19 disease of a country (query Q3), a list of countries is required. The user first needs to display the list of countries on the website (query Q1).

NB: the incidence rate of an epidemic (query Q4)

The incidence rate of an epidemic is called the ratio of the number of new cases during a period over the size of the target population during the same period. It is generally expressed as "the number of new cases per 100,000 people

in a year". This is one of the criteria most important to assess the frequency and the rate of appearance of an epidemic.

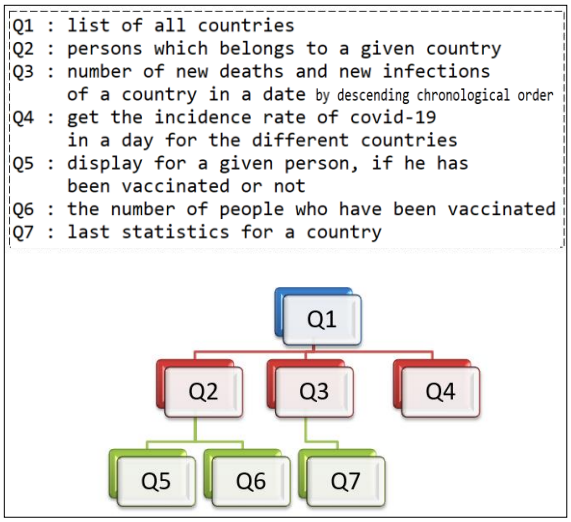


Figure 5. Application workflow

2.3.Logical Data Modeling

The crucial point of the Cassandra data modeling methodology is logical data modeling. It takes a conceptual data model and maps it to a logical data model based on queries defined in an application workflow. A logical data model corresponds to a Cassandra database schema with table schemas defining columns, primary, partition, and clustering keys. We define the query-driven conceptual-to-logical data model mapping via data modeling principles, mapping rules, and mapping patterns.

One of the new features introduced by Cassandra is data nesting. Data nesting refers to a technique that organizes multiple entities (usually of the same type) together based on a known criterion. Such criterion can be that all nested entities must have the same value for some attribute (e.g., persons with the same country) or that all nested entities must be related to a known entity of a different type (e.g., observations that concern a country). Data nesting is used to achieve the "partition per query" access path, such that multiple nested entities can be retrieved from a single partition. There are two mechanisms in Cassandra to nest data: multi-row partitions and collection types. Our methodology primarily relies on multi-row partitions to achieve the best performance. Another key to successful database design is data duplication. Duplicating data in Cassandra across multiple tables, partitions, and rows is a common practice that is required to support efficiently different queries over the same data.

A. Mapping rules

These rules guide a query-driven transition from a conceptual data model to a logical data model.

- MR1 (Entities and Relationships). Entity and relationship types map to tables, while entities and relationships map to table rows. Attribute types that describe entities and relationships at the conceptual level must be preserved as table columns at the logical level. Violation of this rule may lead to data loss.
- MR2 (Equality Search Attributes). Equality search attributes, which are used in a query predicate, map to the prefix columns of a table primary key. Such columns must include all partition key columns and, optionally, one or more clustering key columns. Violation of this rule may result in the inability to support query requirements.
- MR3 (Inequality Search Attributes). An inequality search attribute, which is used in a query predicate, maps to a table clustering key column. In the primary key definition, a column that participates in inequality search must follow columns that participate in an equality search. Violation of this rule may result in the inability to support query requirements.
- MR4 (Ordering Attributes). Ordering attributes, which are specified in a query, map to clustering key columns with ascending or descending clustering order as prescribed by the query. Violation of this rule may result in the inability to support query requirements.
- MR5 (Key Attributes). Key attribute types map to primary key columns. A table that stores entities or relationships as rows must include key attributes that uniquely identify these entities or relationships as part of the table primary key to uniquely identify table rows. Violation of this rule may lead to data loss.

To design a table schema, it is important to apply these mapping rules in the context of a particular query and a subgraph of the conceptual data model that the query deals with. The rules should be applied in the same order as they are listed above.

For example, Figure 6 illustrates how the mapping rules are applied to design a table for query Q3 "number of new deaths and new infections of a country in a date range (ordered by descending chronological order)" that deals with the relationship Country-Concern-Observation (see Figure 4). Figure 6 visualizes a table resulting after each mapping rule, where K and C denote partition and clustering key columns, respectively. The arrows next to the clustering key columns denote ascending (↑) or descending (↓) order. MR1 results in table ObservationsByCountry whose columns correspond to the attribute types used in the query to search for, search on, or order by. MR2 maps the equality search attribute to the partition key column iso. MR3 maps the inequality search attribute to the clustering key column dateObs, and MR4 changes the clustering order to descending. Finally, MR5 maps the key attribute to the clustering key column id.

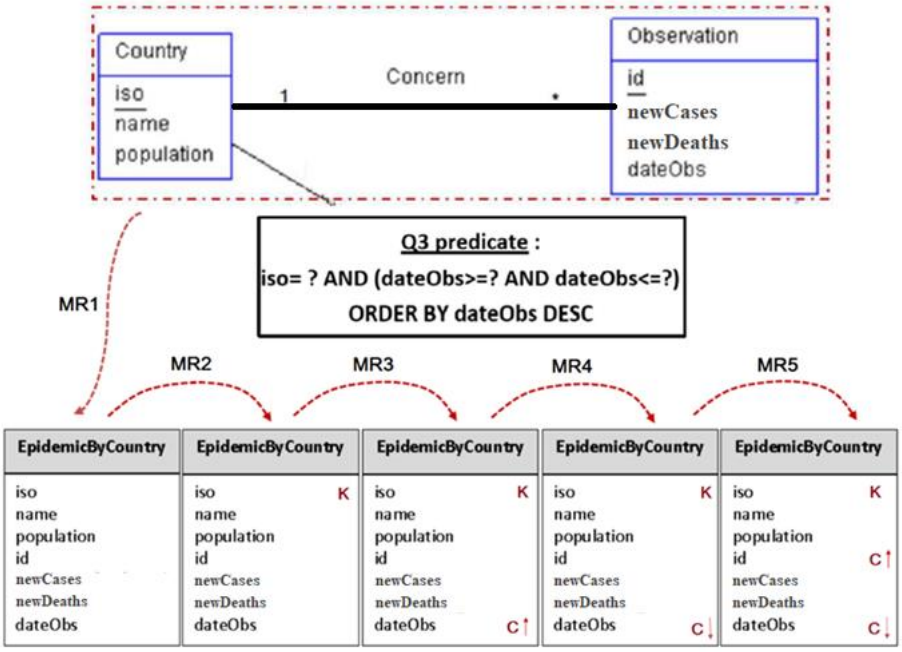


Figure 6. Sample table schema design using the mapping rules

B. Mapping Patterns

Based on the above mapping rules, we design mapping patterns that serve as the basis for automating Cassandra database schema design. Given a query and a conceptual data model subgraph that is relevant to the query, each mapping pattern defines the final table schema design without the need to apply individual mapping rules. While we define a number of different mapping patterns, due to space limitations, we only present one mapping pattern and one example.

A sample mapping pattern is illustrated in Figure7. It is applicable for the case when a given query deals with one-to-many relationships and results in a table schema that nests many entities (rows) under one entity (partition) according to the relationships. When applied to our query "number of new deaths and new infections of a country in a date range (ordered by descending chronological order)" and the relationship Country-Concern-Observation, this mapping pattern results in the table schema shown in Figure6. With our mapping patterns, logical data modeling becomes as simple as finding an appropriate mapping pattern and applying it, which can be automated.

2.4. Physical Data Modeling

The final step of our methodology is the analysis and optimization of a logical data model to produce a physical data model. While the modeling principles, mapping rules, and mapping patterns ensure a correct and efficient logical schema, there are additional efficiency concerns related to database engine constraints or finite cluster resources. A typical analysis of a logical data model involves the estimation of table partition sizes and data duplication factors. Some of the common optimization techniques include partition splitting, inverted indexes, data aggregation, and concurrent data access optimizations. These and other techniques are described in [7].

III. CONCLUSION

In this paper, we have focused on the modeling of oriented-column databases driven by the data and the queries. Our paper was limited to one-to-many associations in the mapping pattern. As a perspective, we can extend our paper to other types of associations. Our paper can be, too, enriched by the development of software to make the process reality and automatically create the database schema by generating the source code in CQL language.

REFERENCES

[1]. Mason, R. T., 'NoSQL databases and data modeling techniques for a document-oriented NoSQL database'. 2015, Proceedings of Informing Science & IT Education Conference (InSITE)

[2]. Mukherjee S., University of the Cumberlands Chicago, United States , The battle between NoSQL Databases and RDBMS'. 2019, Available at SSRN 3393986

[3]. Hanine M., Benderrag A., Boutkhoum O., 'Data Migration Methodology from Relational to NoSQL Databases'. 2015, World Academy of Science, Engineering and Technology International Journal of Computer, Electrical, Automation, Control and Information Engineering, Vol:9, No:12

[4]. Chebotko A., Kashlev A., Lu S., 'A Big Data Modeling Methodology for Apache Cassandra'. 2015, IEEE International Congress on Big Data
[5]. Sam R. Alapati, 'Introduction to the Cassandra Query Language', In book: Expert Apache Cassandra Administration (pp.189-247), January 2018
[6]. Lemahieu W., Vanden B. S., Baesens B., 'Relational Databases: Structured Query Language (SQL)', In book: Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data (pp.146-206), 2019/08/01
[7]. https://www.upsti.fr/espace-etudiants/annales-de-concours/topics/ipt-modelisation-de-la-propagation-d-une-epidemie consulted on June 2021)
[8]. https://docs.datastax.com/en/dse/6.7/cql/cql/ddl/dataModelingApproach.html /(consulted on June 2021)

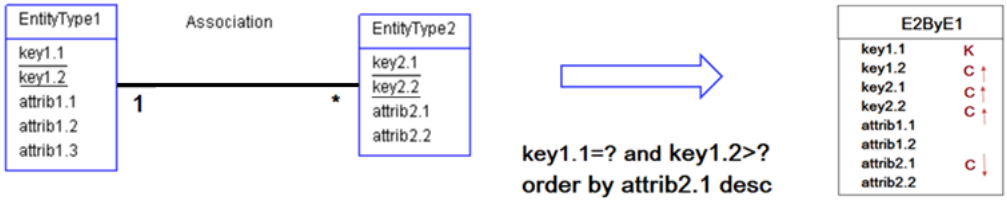


Figure 7. Sample mapping pattern