Advanced Data Management 2017/2018

Team: Andrea Canepa, Stefano Rebora

PART II

OVERVIEW

The reference domain concerns a remote patient monitoring service. In this context, we are interested in two main applications: a mobile application for the patient and a desktop application for the hospital/clinic staff.
Each patient is equipped with one or more medical sensors measuring physiological parameters to perform periodically routine tests and the data are sent to the hospital/clinic database by the mobile application. Data are stored and are not managed as a stream.
The hospital/clinic staff can visualize and analyze the data, write periodically reports that summarize the general patient state of health and suggest a specific treatment.

Typical workloads mix together write and read operations.
The aim of this project is the design and the implementation of the distributed data management layer for the patient mobile application and the staff desktop application. To this aim, we choose Cassandra as reference technology and we rely on CQL for workload implementation.

TECHNOLOGY

As mentioned, we have chosen Cassandra as reference technology. We have already discussed during the course all the main features offered by Cassandra, so in this paragraph we briefly present two important aspects in more details: data replication strategies and lightweight transactions.

Data Replication

Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines the nodes where replicas are placed. The total number of replicas across the cluster is referred to as the replication factor and all replicas are equally important: there is no primary or master replica. Cassandra offers two replication strategies:

- SimpleStrategy
  - Use only for a single datacenter and one rack. SimpleStrategy places the first replica on a node determined by the partitioner. Additional replicas are placed on the next nodes clockwise in the ring without considering topology (rack or datacenter location)

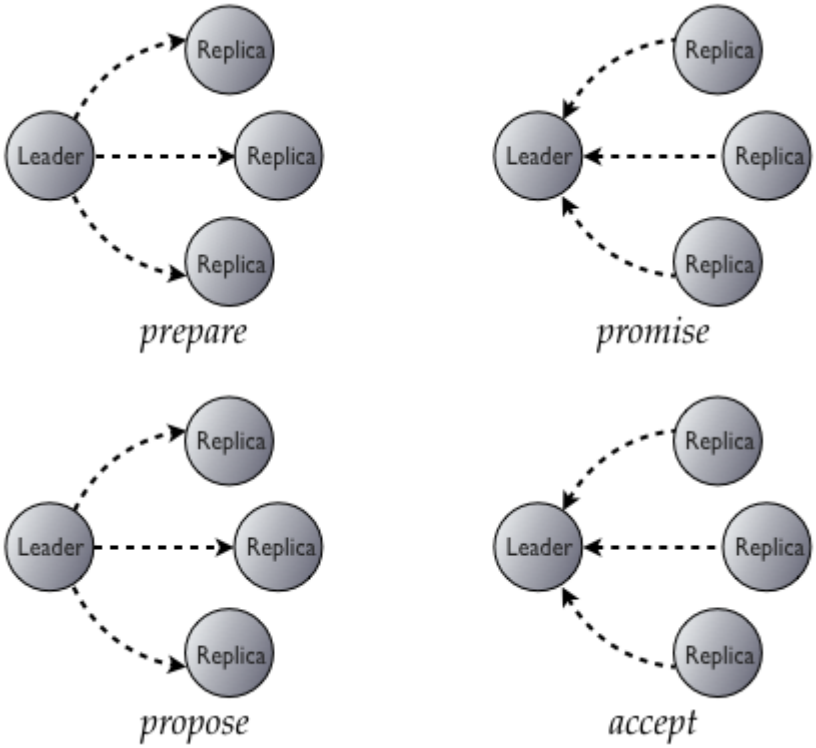
- NetworkTopologyStrategy
  - Use NetworkTopologyStrategy when you have (or plan to have) your cluster deployed across multiple datacenters.This strategy specifies how many replicas you want in each datacenter. NetworkTopologyStrategy places replicas in the same data center by walking the ring clockwise until reaching the first node in another rack. NetworkTopologyStrategy attempts to place replicas on distinct racks because nodes in the same rack (or similar physical grouping) often fail at the same time due to power, cooling, or network issues.

For our project's purposes, the first strategy in enough considering that our cluster is executed in a pseudo-distributed execution mode and is made up by only one rack.

Lightweight transactions

Lightweight transactions with linearizable consistency ensure transaction isolation level similar to the serializable level offered by RDBMS's. They are also known as compare and set transactions. We use lightweight transactions instead of durable transactions with eventual/tunable consistency for situations the require nodes in the distributed system to agree on changes to data. For example, two users attempting to create a unique user account in the same cluster could overwrite each other's work. Using a lightweight transaction, the nodes can agree to create only one account.
Cassandra implements lightweight transactions by extending the Paxos consensus protocol, which is based on a quorum-based algorithm.

The Paxos consensus protocol allows a distributed system to agree on proposals with a quorum-based algorithm, with no masters required and without the problems of two-phase commit. There are two phases to Paxos: prepare/promise, and propose/accept.

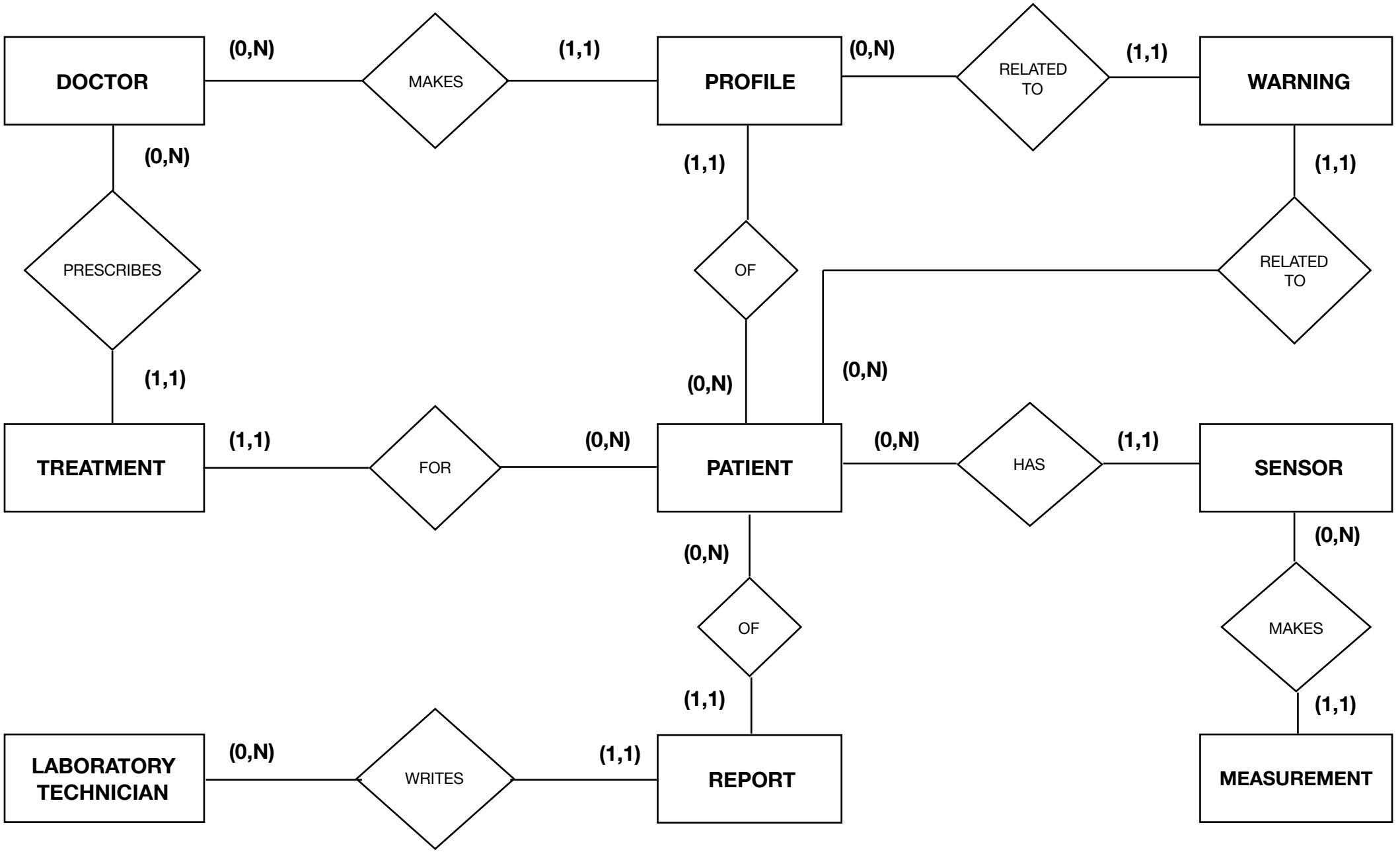


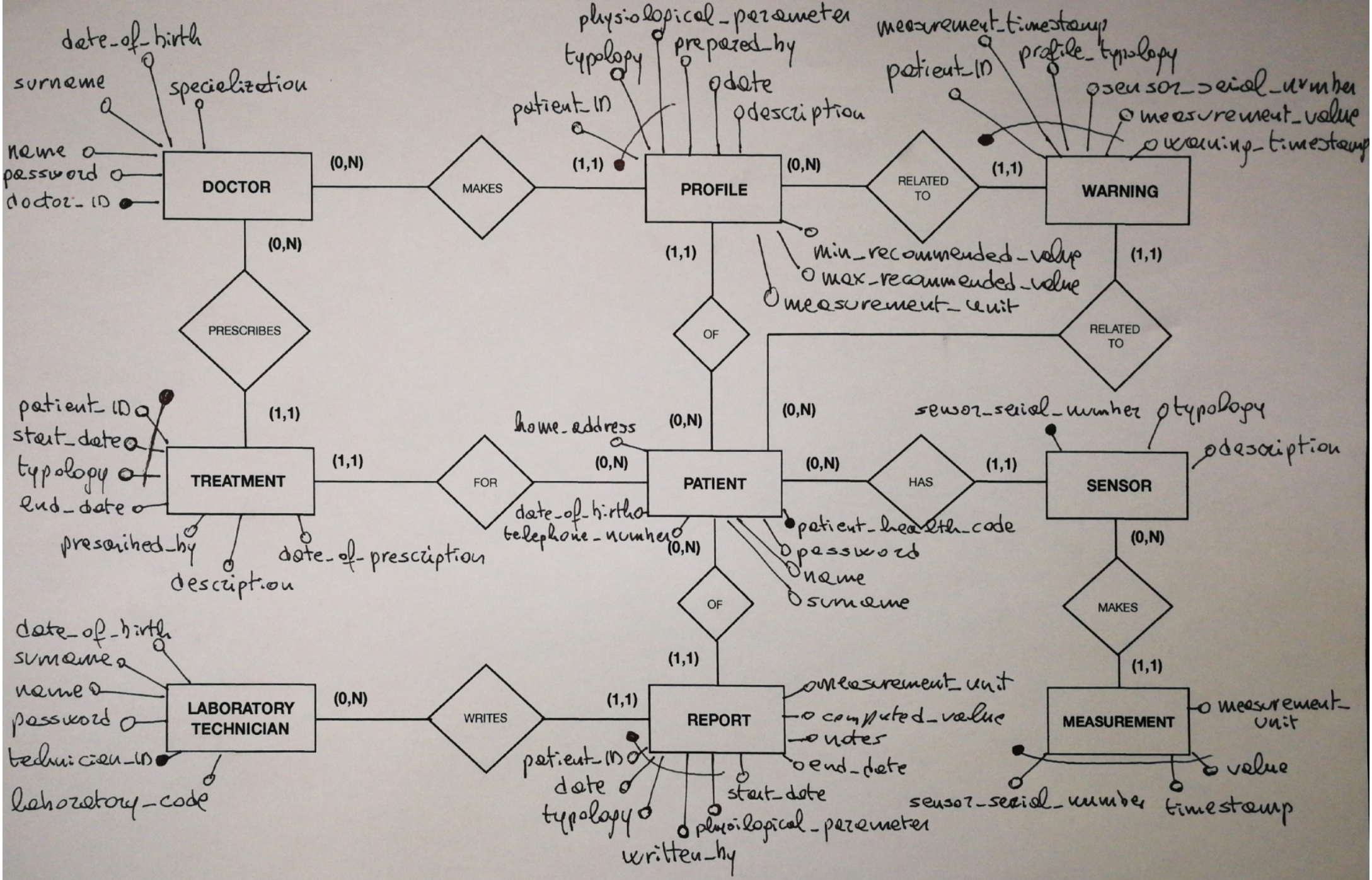
Prepare/promise is the core of the algorithm. Any node may propose a value; we call that node the leader. (Note that many nodes may attempt to act as leaders simultaneously. This is not a "master" role.)
The leader picks a ballot and sends it to the participating replicas. If the ballot is the highest a replica has seen, it promises to not accept any proposals associated with any earlier ballot. Along with that promise, it includes the most recent proposal it has already received.
If a majority of the nodes promise to accept the leader's proposal, it may proceed to the actual proposal, but with the wrinkle that if a majority of replicas included an earlier proposal with their promise, then that is the value the leader must propose. Conceptually, if a leader interrupts an earlier leader, it must first finish that leader's proposal before proceeding with its own, thus giving us our desired linearizable behaviour.

For our project, we have applied the lightweight transactions in order to guarantee a unique user account creation.

LOGICAL SCHEMA

The following diagrams are two ER diagrams modelling our system's entities:


DOCTOR
(0,N)
MAKES
(1,1)
PROFILE
(0,N)
RELATED TO
(1,1)
WARNING
(0,N)
PRESCRIBES
(1,1)
(1,1)
OF
(0,N)
RELATED TO
(1,1)
(0,N)
TREATMENT
(1,1)
FOR
(0,N)
PATIENT
(0,N)
HAS
(1,1)
SENSOR
(0,N)
(0,N)
OF
MAKES
(1,1)
(1,1)
LABORATORY TECHNICIAN
(0,N)
WRITES
(1,1)
REPORT
MEASUREMENT


DOCTOR
name
password
doctor_ID
surname
specialization
date_of_birth
(0,N)
MAKES
(1,1)
PROFILE
typology
patient_ID
physiological_parameter
prepared_by
date
description
(0,N)
RELATED TO
(1,1)
WARNING
measurement_timestamp
patient_ID
profile_typology
sensor_serial_number
measurement_value
warning_timestamp
(0,N)
PRESCRIBES
(1,1)
(1,1)
OF
min_recommended_value
max_recommended_value
measurement_unit
RELATED TO
(1,1)
TREATMENT
patient_ID
start_date
typology
end_date
prescribed_by
description
date_of_prescription
(1,1)
FOR
(0,N)
PATIENT
home_address
date_of_birth
telephone_number
(0,N)
(0,N)
HAS
(1,1)
SENSOR
sensor_serial_number
typology
description
patient_healthcode
password
name
surname
(0,N)
MAKES
(1,1)
OF
(1,1)
LABORATORY TECHNICIAN
date_of_birth
surname
name
password
technician_ID
laboratory_code
(0,N)
WRITES
(1,1)
REPORT
measurement_unit
computed_value
notes
end_date
patient_ID
date
typology
start_date
physiological_parameter
written_by
MEASUREMENT
measurement_unit
value
timestamp
sensor_serial_number

WORKLOAD

Typical workloads mix together write and read operations. The following list is a list of typical and frequent operations broken down by system users. The most frequent operations are underlined in green.

Doctors

- Given all the doctor's information, create a unique account for a doctor (the doctor ID can be assigned by the system) (or remove)
- Log in, given the doctor ID and the current password
- Change the password, given the doctor ID, the old password and the new one
- Given the doctor ID, visualize the doctor's account information
- Given a patient ID, visualize the patient's account information
- Prescribe a treatment, given a patient ID
- Modify a treatment prescribed by the doctor for a given patient ID (or remove)
- Given a patient ID, get the list of all treatments prescribed to the patient by the doctor in a given time period
- Given a patient ID, get the list of all treatments that the patient has to follow in a given time period
- Given a patient ID, get the list of all treatments of a specific typology prescribed to the patient in a given time period
- Make a profile for a patient
- Modify a patient's profile made by the doctor (or remove)
- Get the list of all profiles of a patient (both information and values, see the relevant design details section)
- Get the list of warnings of a given patient in a given time period
- Get the list of reports of a given patient in a given time period (both information and values)

Patients

- Given all the patient's information, create a unique account for a patient (or remove)
- Log in, given the patient ID and the current password
- Change the password, given the patient ID, the old password and the new one
- Given the patient ID, visualize the patient's account information
- Given the patient ID, get the list of the treatments that the patient has to follow in a given time period
- Given the patient ID, get the list of the reports in a given time period (both information and values)
- Given the patient ID, get the list of all profiles (both information and values)
- Given the patient ID, associate a new sensor (given all the information about the sensor) to the patient's account (e.g. scanning a QR code with the mobile application) or remove a sensor
- Given the patient ID, get the list of all sensors connected to the patient

Laboratory Technicians

- Given all the laboratory technician's information, create a unique account for the lab. technician (the technician ID can be assigned by the system) (or remove)
- Log in, given the technician ID and the current password
- Change the password, given the technician ID, the old password and the new one
- Given the technician ID, visualize the technician's account information
- Given a patient ID, visualize the patient's account information
- Get the list of all profiles of a patient
- Get the list of warnings of a given patient in a given time period
- Make a new report for a patient
- Modify a report for a patient (or remove)
- Get the list of reports of a given patient in a given time period (both information and values)
- Given the patient ID, get the list of all sensors connected to the patient
- Given the patient ID, get the list of all sensors connected to the patient of a given typology
- Given a sensor ID, get the list of all sensor's measurements in a given time period
- Given a sensor ID, compute an aggregate value for the sensor's measurements in a given time period (max value, min value, avg value)

Sensors

- Given the sensor ID, the value and the unit of the measurement and the measurement 's time stamp, insert a new measurement

Warning Daemon

An application process automatically checks if a new measurement of a patient respects the recommended values specified by the patient's profiles. We assume that this warning daemon periodically gets for a patient the profile list and inserts a new warning if a measurement value is anomalous for a profile. The daemon frequently performs the following operations on the cluster:

- Given a sensor ID, get the associated patient ID
- Given a patient ID, get the list of all patient's profiles (only recommended values)
- Given a patient's profile and a measurement, insert a new warning
- Given a patient ID, remove all patient's warnings in a given time period (periodical garbage collector)

RELEVANT DESIGN DETAILS

Considering the conceptual schema, the defined workload and the features provided by Cassandra, we have taken the following design decisions for the logical schema:

- Separation between profile information and profile recommended values in order to avoid unnecessary data replication and take advantage of dynamic columns
- Separation between report information and report values in order to avoid unnecessary data replication and take advantage of dynamic columns
- Creation of the patientSensors table in order to separate the information about the link between a patient and his sensor from the sensor itself. With this approach, each sensor has his own partition with his measurements.
- Assumption that the application level takes care of the overlapping constraint for a treatment of a given patient and a given typology before the insert of a new treatment

LOGICAL SCHEMA

KEY:
Primary key
Partition key
Clustering column

Doctor (doctor_ID, password, name, surname, date_of_birth, specialization)

LaboratoryTechnician (technician_ID, password, name, surname, date_of_birth, laboratory_code)

Treatment (patient_ID, start_date, typology, end_date, prescribed_by, description, date_of_prescription)

ProfileInformation(patient_ID, typology, prepared_by, date, description)

ProfileRecommendedValues(patient_ID, typology, physiological_parameter, min_recommended_value, max_recommended_value, measurement_unit)

Patient (patient_health_code,password, name, surname, date_of_birth, telephone_number, home_address)

PatientSensors(patient_ID, sensor_typology, sensor_serial_number)

Warning (patient_ID, measurement_timestamp, profile_typology, sensor_serial_number,measurement_value, warning_timestamp)

ReportInformation(patient_ID, date, typology, written_by, start_date, end_date, notes)
Note: date is represented as timestamp

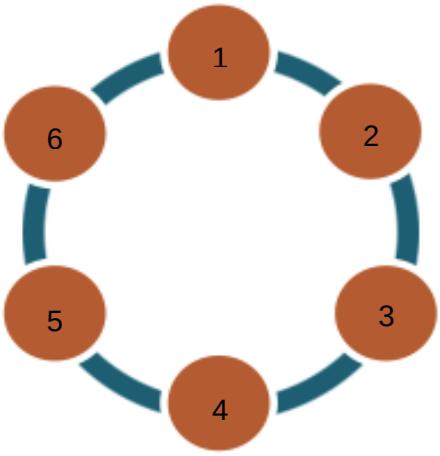
ReportValues (patient_ID, date, typology, written_by, physiological_parameter, computed_value, measurement_unit)

Sensor (sensor_serial_number, typology, description)

Measurement (sensor_serial_number, timestamp, value, measurement_unit)

CLUSTER SET UP

To create a cluster for our project, we have relied on [Cassandra Cluster Manager] that makes easy to create, manage and destroy a small Cassandra cluster on a local box.
We have chosen to create a cluster of six nodes and one rack, with a replication factor of three and a replication strategy ‘SimpleStrategy’ as already specified in first paragraph : in this way, the 50% of the data is owned by each node.



Our cluster is based on Cassandra ring topology and distributed hash partitioning

```
Datacenter: datacenter1
==========
Address    Rack        Status State   Load            Owns                Token
                                                                           6148914691236517202
127.0.0.1  rack1       Up     Normal  97.85 KiB       50.00%              -9223372036854775808
127.0.0.2  rack1       Up     Normal  100.86 KiB      50.00%              -6148914691236517206
127.0.0.3  rack1       Up     Normal  100.88 KiB      50.00%              -3074457345618258604
127.0.0.4  rack1       Up     Normal  100.83 KiB      50.00%              -2
127.0.0.5  rack1       Up     Normal  97.86 KiB       50.00%              3074457345618258600
127.0.0.6  rack1       Up     Normal  97.85 KiB       50.00%              6148914691236517202
```

Then we have relied on Cassandra’s tunable consistency for client request. We have applied a CONSISTENCY LOCAL_QUORUM for the workload operations that require a strong consistency and a CONSISTENCY ONE for operations where consistency requirements are not stringent.