Prototipo Prova Scritta

today = giorno della prova scritta

today = "08/07/2024" no variabili Today Tomorrow Yesterday ecc.

Name Surname inverit. query 1

query 2 Pat è fissato
Tra Patient e Visit

Query 1 : diff. media alta

Query 2 : diff. semplice

Quindi NO esistenziale senza nessuna variabile
può benissimo essere che ho già
fatto le variabili

..... ∧ Calendar(Day) ∧ .....

Sempre disponibile, mi
dice che voglio sapere che
assumono la data del calendario

Query c.

Faccio query come per trovare che esista quella cosa (ragiono in positivo)
e poi nego l'esistenziale esterno

Query d.

∀ ⇒

Va sempre l'implica

React Property

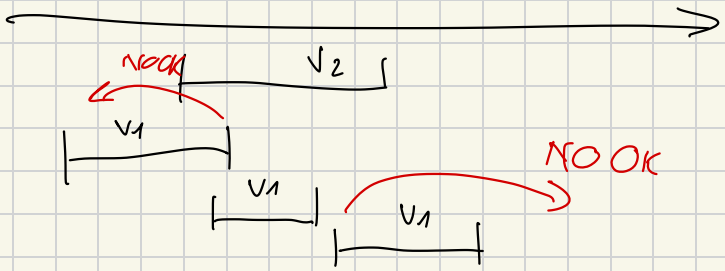
⋮

patient — è chiaro che ci va la chiave del paziente ma non posso specificarlo

Se voglio accedere pure VDa Patient a Reazioni

devo specificare tutte le reazioni associate al paziente

4a.



OK

Addiacenza Spaziale → Touches

18 giugno 2021

1.a $\{ D.name, D.Surname \mid \exists IDcode, dep, Hosp (Doctor(DCode, D.Name, D.Surname, Dep, Hosp) \wedge$

$\neg \exists Date, Pat, Type, Time (Surgery (DCode, Date, Pat, Type, Time)$

$\wedge (Date \leq '05/06/2020' \wedge Date \geq '01/06/2020') \wedge$

$\exists P.Surname, P.name, P.Region (PATIENT (Pat, P.Surname, P.name, 'Venice', P.Region))) \}$

1.b

$\{ P.name, P.Surname \mid \exists P.code, P.res (PATIENT (P.code, P.Surname, P.name, P.res, 'Lombardy') \wedge$

$\exists H.start, H.Dep, H.Hosp, H.End (HOSP(H.Start, P.code, H.Dep, H.hosp, H.End) \wedge$

$(H.Start \geq '1/02/2020' \wedge H.Start \leq '29/02/2020') \wedge$

$\exists DepChief, Address, Mun(DEPARTMENT(H.Dep, H.Hosp, Address, Mun) \wedge Mun = 'Verona'$

$\wedge \exists Doctor, Date, Time (SURGEY (Doctor, Date, P.code, 'S1', Time) \wedge$

$Date < H.End)))) \}$

1.c

$\{ D.Name, D.Hosp, D.Mun \mid \exists D.Chief, D.Addr (DEPARTMENT (D.name, D.hosp, D.Chief, D.Addr, D.Mun)$

$\wedge \exists H.Start1, H.Pat1, H.End1 (HOSP (H.Start1, H.Pat1, D.name, D.hosp, H.End1)$

$\wedge H.Start1 = '1/10/2021'$

$\wedge \exists H.Start2, H.Pat2, H.End2 (HOSP (H.Start2, H.Pat2, D.name, D.hosp, H.End2)$

$\wedge H.Start2 = '1/10/2021' \wedge H.Pat1 \neq H.Pat2$

$\wedge \exists P.Surname1, P.Name1, P.Reg1 (PATIENT(H.Pat1, P.Surname1, P.Name1, 'Rome', P.Reg1)$

$\wedge \exists P.Surname2, P.Name2, P.Reg2 (PATIENT(H.Pat2, P.Surname2, P.Name2, 'Rome', P.Reg2)))))) \}$

1.d

$\{ S1.type \mid \forall D.Doc, D.Name, D.Surname, D.Dep, D.hosp (Doc(D.Doc, D.Name, D.Surname, D.Dep, Dhosp))$

$\rightarrow \exists S1.Date, S1.Pat, S1.time (Surgey (D.Doc, S1.Date, S1.Pat, S.type,$

$S.time))) \wedge S.Date \geq '1/03/2020' \wedge S.Date \leq '31/03/2020' \}$

```
SELECT  Doc.name, Doc.Surname
FROM  Patient P UNNEST  P.Hosp Hosp UNNEST Hosp.Surgery Su
      JOIN  DEPARTMENT Dep UNNEST Dep.DOCTORS Doc  ON KEYS  Su.Doctor_id
      WHERE  D.id NOT IN  ( SELECT  S.Doctor.id
                            FROM  Patient P1 UNNEST  P1.Hosp H  UNNEST H.Surgery Su1
                            WHERE Su1.DATE ≥ '1/06/2020' ∧ Su1.DATE ≤ '05/06/2020'
                            ∧ P1.restown = 'Venice' )
```

1.b.

```
SELECT  P.name, P.Surname
FROM  PATIENT P UNNEST P.Hosp H  UNNEST H.SURGERY Su
      JOIN  DEPARTMENT D  ON KEYS  Hosp.DEPARTMENT.ID
      WHERE  D.Municipality = 'Varese' ∧ HOSP.StartDate ≥ '1/02/2020' ∧
             HOSP.StartDATE ≤ '29/02/2020'  ∧ P.Region = 'Lombardy'
             ∧  Su.Date ≤  Hosp.EndDate  ∧  Su.Type = 'S1'
```

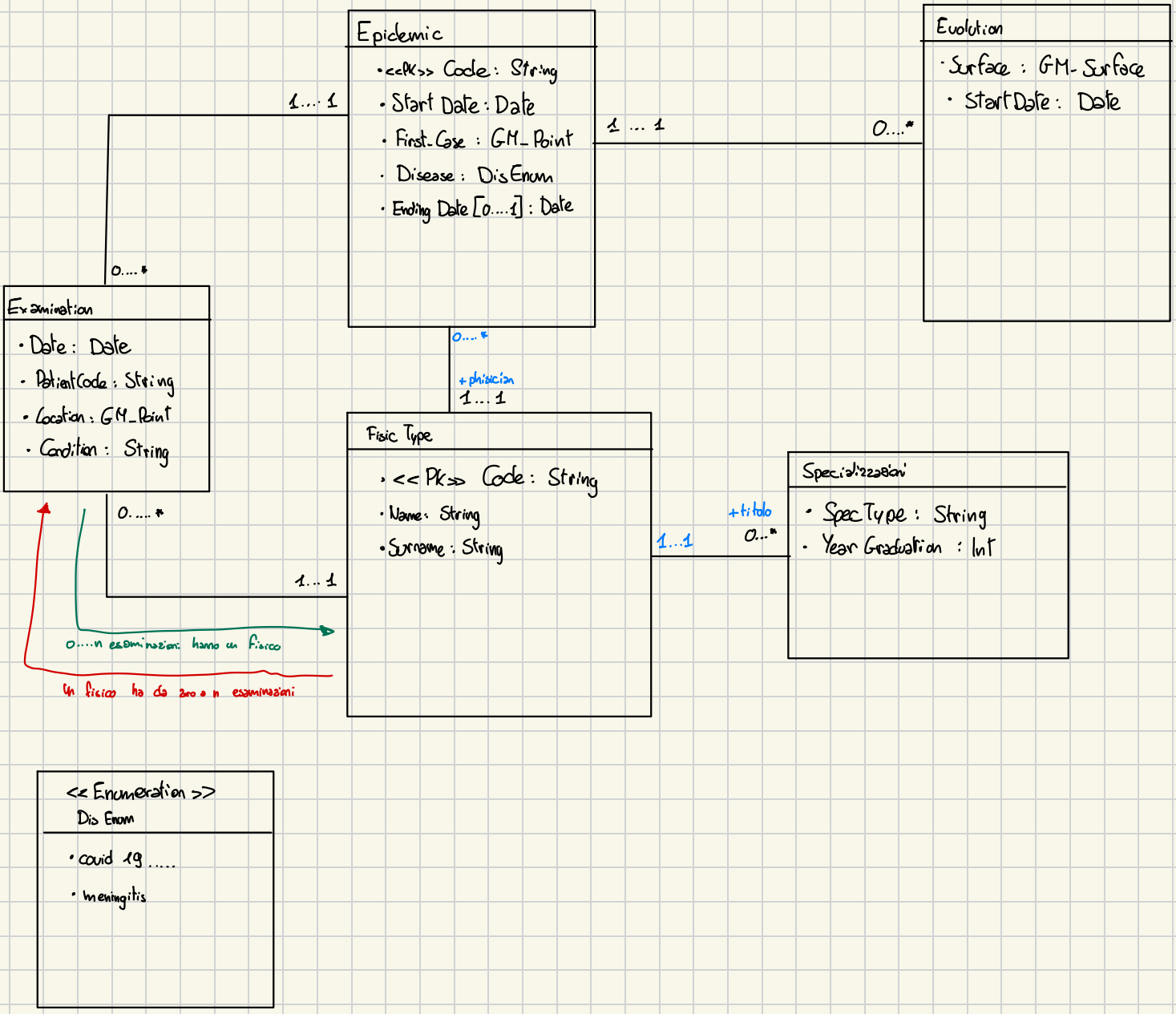
1.c.

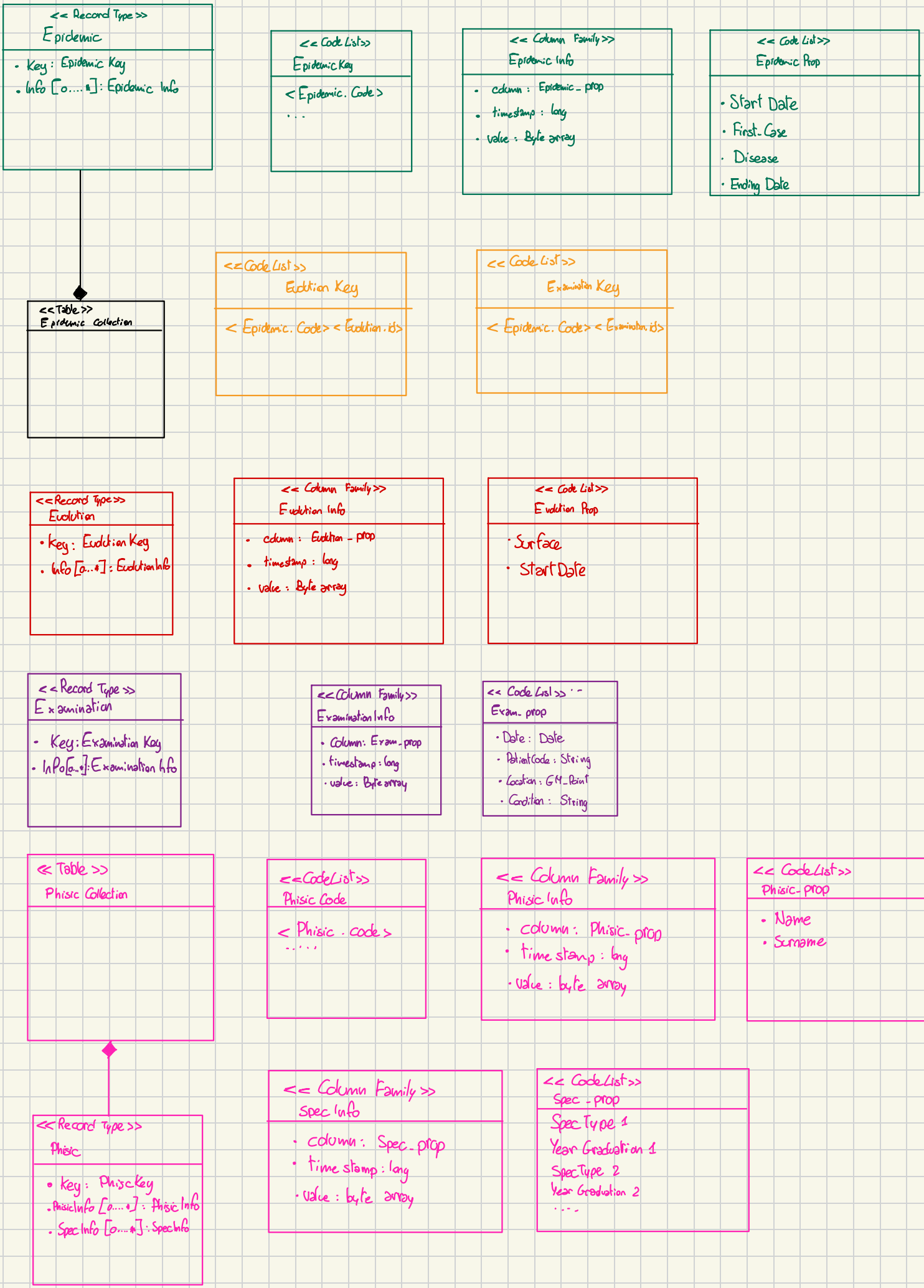
```
SELECT  D.name, D.hospital, D.municipality
FROM  DEPARTMENT  D
      JOIN  PATIENT P1 UNNEST P1.Hosp H1  ON  H1.DEPARTMENT_ID = D.ID
      JOIN  PATIENT P2 UNNEST P2.Hosp H2  ON  H2.DEPARTMENT_ID = D.ID
WHERE  P1.Code ≠ P2.Code  ∧  P1.ResTown = 'Rome' ∧ P2.ResTown = 'Rome'
       ∧ H1.StartDate = '1/10/2021'  ∧  H2.StartDate = '1/10/2021'
```

```
SELECT  SU.Type
FROM    Patient P UNNEST P.Hospitalization H UNNEST H.Surgery SU
WHERE   Date > 1 Marzo ∧ Date < 1 Marzo
Group BY S.type
HAVING  COUNT (DISTINCT S.Doctor_id) = (SELECT COUNT(*)
                                        FROM DOCTOR)
```

$$\{ S.type \mid \exists D.code, S.Date, S.Pat, S.type, S.time \left(Surgery(D.code, S.date, S.Pat, S.type, S.time) \right.$$
$$\wedge \forall D.Name, D.Surname, D.Dep, D.hospital \left(D.code, D.name, D.Surname, D.Dep, \right.$$
$$\left. \left. D.hospital \right) \rightarrow S.Date > '1/3/2024' \wedge S.Date < '31/3/2024' \right\}$$

2. Given the following requirements, design a database at conceptual level using UML and translate the class diagram into a physical data model for the HBase system.
The considered information system deals with the evolution of the epidemics on the territory of the Veneto region. For each epidemic the system stores: a unique code, the starting date, the point representing the first case detection, the disease (from a predefined list: covid19, meningitis, …), the ending date (optional) and the physician which followed or is following the epidemic evolution. For every epidemic its evolution is stored by means of a sequence of records containing: the surface (spatial attribute), representing the extensions of the epidemic on the territory, and the date when the epidemic started to have the specified extension. For each physician: a unique code, the name and surname of the physician and the list of her/his specializations (described by: specialization type and year of graduation) is stored. During an epidemic some examinations regarding a specific patient can be performed, in this case the system stores: the date of the examination, the code of the patient, the location of the patient by mean of a point (spatial attribute), the physician and the annotation describing the patient condition.




<< Record Type >>
Epidemic
• Key : Epidemic Key
• Info [0....*] : Epidemic Info
<< Code List >>
Epidemic Key
< Epidemic. Code >
...
<< Column Family >>
Epidemic Info
• column : Epidemic_prop
• timestamp : long
• value : Byte array
<< Code List >>
Epidemic Prop
• Start Date
• First_Case
• Disease
• Ending Date
<< Table >>
Epidemic Collection
<< Code List >>
Evolution Key
< Epidemic. Code > < Evolution. id >
<< Code List >>
Examination Key
< Epidemic. Code > < Examination. id >
<< Record Type >>
Evolution
• key : Evolution Key
• Info [0...*] : Evolution Info
<< Column Family >>
Evolution Info
• column : Evolution_prop
• timestamp : long
• value : Byte array
<< Code List >>
Evolution Prop
• Surface
• Start Date
<< Record Type >>
Examination
• Key : Examination Key
• InPo [0..*] : Examination Info
<< Column Family >>
Examination Info
• Column : Exam_prop
• timestamp : long
• value : Byte array
<< Code List >>
Exam_prop
• Date : Date
• PatientCode : String
• Location : GM_Point
• Condition : String
<< Table >>
Phisic Collection
<< Code List >>
Phisic Code
< Phisic . code >
...
<< Column Family >>
Phisic Info
• column : Phisic_prop
• time stamp : long
• value : byte array
<< Code List >>
Phisic_prop
• Name
• Surname
<< Record Type >>
Phisic
• key : Phisc key
• PhisicInfo [0....*] : Phisic Info
• SpecInfo [0....*] : SpecInfo
<< Column Family >>
Spec Info
• column : Spec_prop
• time stamp : long
• value : byte array
<< Code List >>
Spec_prop
Spec Type 1
Year Graduation 1
Spec Type 2
Year Graduation 2
...

3. $F = \{ AC \to D,\ A \to B,\ BC \to DR,\ D \to P,\ P \to S,\ ST \to PQ \}$

Scomposizione BCNF

Calcolo chiave

1 riscrivere come relazione le dipendenze

$R_1 (A, C, D)$

$R_2 (A, B)$

$R_3 (B, C, D, R)$

$R_4 (D, P)$

$R_5 (P, S)$ ← rimuovo

$R_6 (S, T, P, Q)$

2 Aggiungi relazione con la chiave

$R_7 (A, C, T)$

3 Rimuovi cose superflue

Test BCNF

1 Per ogni relazione prendo ogni attributo dello schema e calcolo la chiusura

2 Controllo se lo schema è sottoinsieme della chiusura. Se sì prox. attributo

3 Se no controllo che Chiusura $\cap$ (Schema − Attributo) $= \emptyset$ Se sì prox attr. Se no termino

Test 3NF

1 Per ogni relazione calcolo la chiave guardando F

2 Per ogni attr. della relazione calcolo chiusura

3 Controllo se lo schema è sottoinsieme della chiusura. Se sì prox. attributo

4 Se no controllo che Chiusura $\cap$ (Schema − Attributo) $= \emptyset$

5 Chiusura $\cap$ (Schema − Attributo)

3. $F = \{ AC \to D,\ A \to B,\ BC \to DR,\ D \to P,\ P \to S,\ ST \to RQ \}$

$R_1 = (A, B, C, D)$

$attr = \{A\}$

$attr^+ = \{A, B\}$

$\{A, B, C, D\} \subseteq \{A, B\} =$ false

$\{A, B\} \cap (\{A, B, C, D\} - \{A\}) \neq \emptyset$ false

lo schema non è in BCNF quindi ha ridondanza

$F = \{ A \to BC,\ ST \to W,\ W \to T \}$

$R_1 (A, B, C, D)$ $R_2 (S, T, W)$

$A^+ = \{A, B, C\}$

$\{A, B, C, D\} \subseteq \{A, B, C\} =$ False ×

$\{(A, B, C)\} \cap \{(A, B, C, D) - (A)\} \neq \emptyset$ ×

In R_1 la ridondanza è B, C

facciamo $R_2 (S, T, W)$

$S^+ = \{S\}$

$\{S, T, W\} \subseteq \{S\}$ ×

$\{S\} \cap (\{S, T, W\} - \{S\}) = \emptyset$ ✓

$T^+ = \{T\}$

$\{S, T, W\} \subseteq \{T\}$ ×

$\{T\} \cap (\{S, T, W\} - \{T\}) = \emptyset$ ✓

$W^+ = \{W, T\}$

$\{S, T, W\} \subseteq \{W, T\}$ ×

$\{W, T\} \cap (\{ST, W\} - \{W\}) \neq \emptyset$ ✓

C'è ridondanza su T

1 Per ogni dipendenza funzionale crea una relazione se con gli att della dipendenza se non esiste già

2 Calcola una chiave K rispetto ad F

3 Se non esiste una relazione j t.c. $K \subseteq R_j$

4 Aggiungiamo una relazione con la chiave

Schema $\subseteq$ Chiusura ? Se sì next att

Chiusura $\subseteq$ (Schema - Att) Se sì next att
Se' no forse

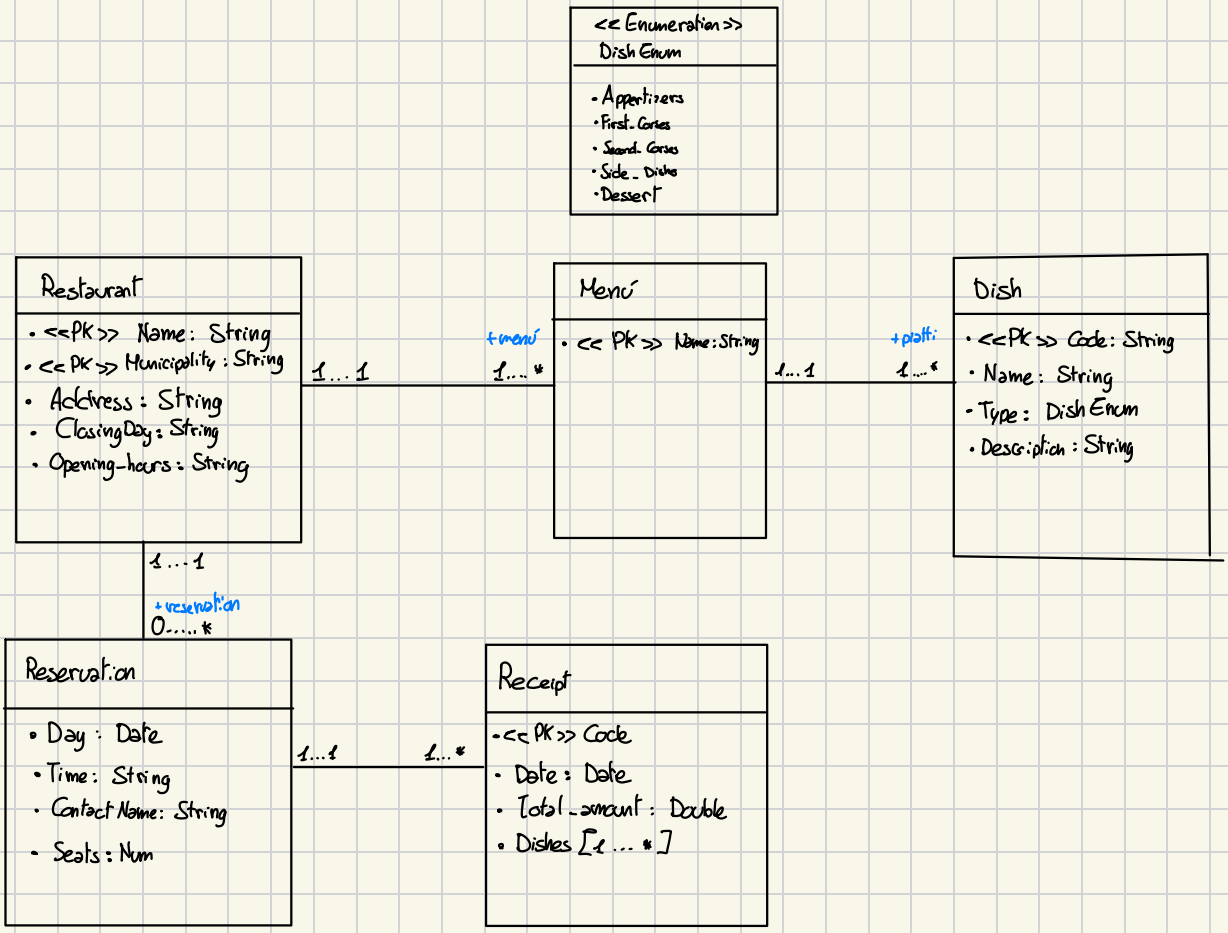
SELECT E1.Code, E2.Code, E1.Disease
FROM EPIDEMIC E1, EPIDEMIC E2, MUNICIPALITIES M, EPIDEMIC_EVOLUTION EV1, EPIDEMIC_EVOLUTION EV2
WHERE E1.START_VT ≥ '1.1.2020' ∧ E1.START_VT ≤ '31.12.2020' ∧ E1.END > E2.START ∧ E1.Code ≠ E2.Code
∧ E1.DISEASE = E2.DISEASE ∧ EV1.CodeEpidemic = E1.Code ∧ EV2.CodeEpidemic = E2.Code ∧
∧ NOT ST_DISJOINT (EV1.Extension, EV2.Extension)
∧ (EV1.Start_VT, EV1.End_VT : 1) OVERLAP (EV2.Start_VT, EV2.Start_VT)
∧ EV2.Start_VT = E2.Start_VT
∧ M.name = 'Venice' ∧ NOT ST_DISJOINT (EV1.extension, M.extension)

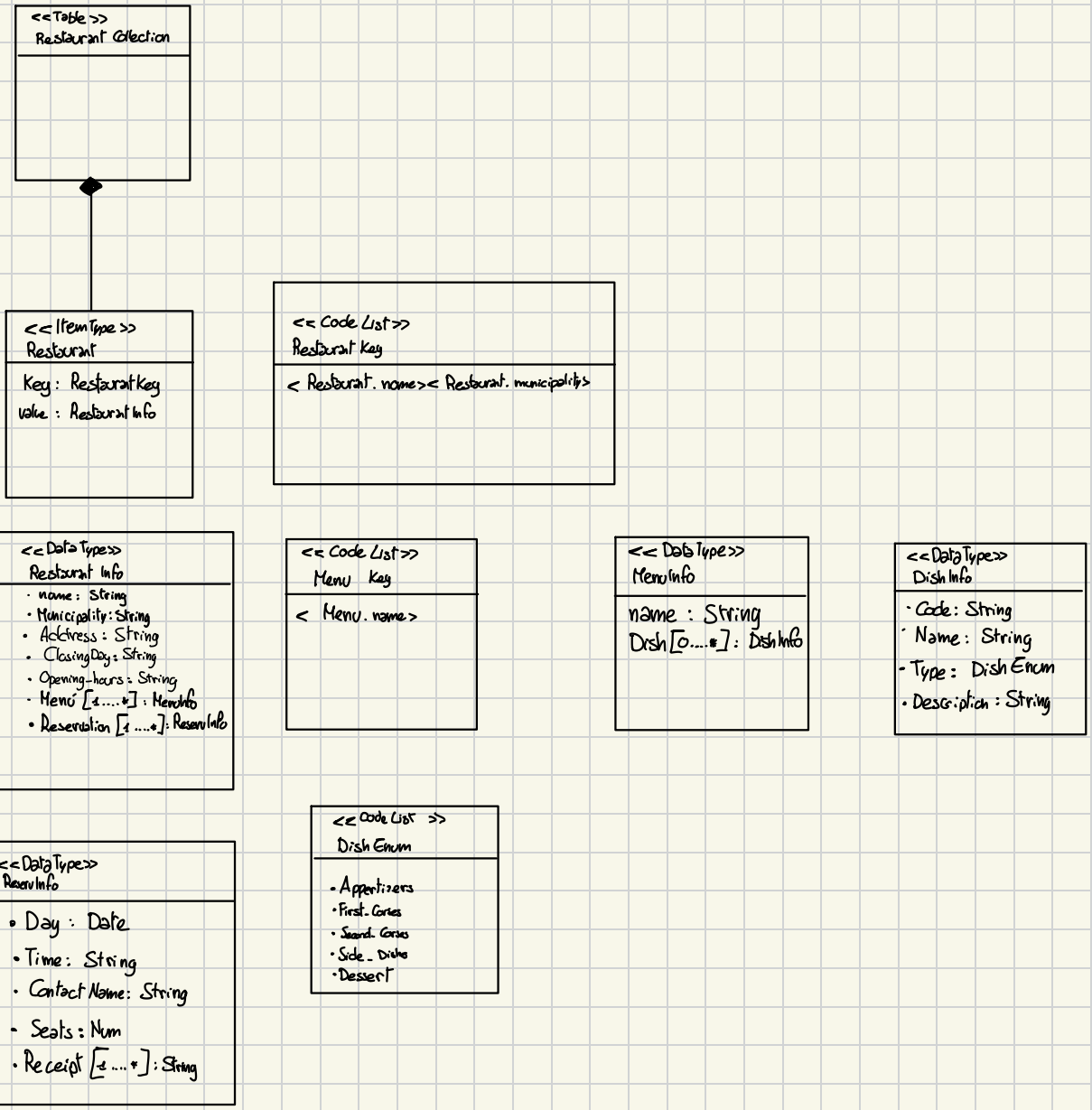
(b)

SELECT M.Code, M.name, COUNT (*)
FROM Municipality M, Epidemic E
WHERE M.Inhabitants < 2500 ∧ ST_CONTAINS (M.EXTENSION, E.Point First Case)
∧ E.START_VT ≥ '1/1/2021' ∧ E.START_VT ≤ '31/12/2021'
GROUP BY M.Code, M.Name

2. Given the following requirements, design a database at conceptual level using UML and translate the class diagram into a physical data model for the DynamoDB system.
The considered information system deals with a set of restaurants. For each restaurant, the system stores: the name, the municipality in which it is located (the couple [name, municipality] uniquely identifies a restaurant), the address, the closing day and the opening hours. In addition, for each restaurant the menus it offers are recorded.
Each menu is characterized by a unique name and a list of dishes offered: one or more appetizers, one or more first courses, one or more second courses, one or more side dishes and one or more desserts. For each dish you register: unique code, name, type (appetizer, first course, second course, etc. ...) and a description.
The system also manages customer reservations for restaurants by recording in addition to the restaurant: the day, the time, the contact name and the number of booked seats.
For each meal, the system records the receipt by storing: the progressive number of the receipt (unique), the date and the total amount. Furthermore, for each receipt, the dishes that have been served are recorded (for example receipt 107 could contain: 3 mushroom risottos, 4 cutlets, 3 salads and 2 ice creams).

2.a (5) Design the UML class diagram at conceptual level
2.b (5) Generate the corresponding physical schema for DynamoDB considering as main access paths: the restaurant, containing all the information describing the menus offered by the restaurant, and a collection containing the reservation and the receipt of each day of the year.




<<Table>>
Restaurant Collection
<<ItemType>>
Restaurant
Key : RestaurantKey
value : RestaurantInfo
<<Code List>>
RestaurantKey
< Restaurant. name>< Restaurant. municipality>
<<DataType>>
RestaurantInfo
· name : String
· Municipality : String
· Address : String
· ClosingDay : String
· Opening-hours : String
· Menù [1....*] : MenuInfo
· Reservation [1 ....*]: ReservInfo
<<Code List>>
Menu Key
< Menu. name>
<<DataType>>
MenuInfo
name : String
Dish[0....*] : DishInfo
<<DataType>>
DishInfo
· Code : String
· Name : String
· Type : Dish Enum
· Description : String
<<DataType>>
ReservInfo
• Day : Date
• Time : String
• Contact Name : String
• Seats : Num
• Receipt [1 ....*] : String
<<Code List>>
Dish Enum
• Appetizers
• First_Courses
• Second_Courses
• Side_Dishes
• Dessert

1) $\{ D.name, D.Surname \mid \exists D.Code, D.Dep, D.Hosp \ (Doctor(D.Code, D.Name, D.Surname, D.Dep, D.Hosp)$

$\wedge \exists Dep.Name, Dep.Hosp, Dep.Add, Dep.Mun \ (Department(Dep.Name, Dep.Hosp, D.Code, Dep.Add, Dep.Mun)$

$\wedge \neg \exists H.Start, H.Pat, H.End \ (Hosp(H.Start, H.Pat, Dep.Name, Dep.Hosp, H.End) \wedge$

$\exists P.age, P.Region(PATIENT(H.Pat, P.age, "Rome", P.Region) \wedge NOT(H.Start > '31/08/2021' \wedge H.End < '1/08/2021'))))\}$

2) $\{ P.Code, P.Age \mid \exists P.ResTown, (PATIENT(P.Code, P.Age, P.ResTown, 'Lombardy') \wedge$

$\exists H.Start, H.Dep, H.Hosp, H.End \ (HOSP(H.Start, P.Code, H.Dep, H.Hosp, H.End) \wedge$

$\exists Dep.DepChief, Dep.Address \ (H.Dep, H.Hosp, Dep.DepChief, Dep.Address, 'Verona') \wedge H.Start \geq '1/07/2021' \wedge$

$H.Start \leq '31/07/2021' \wedge$

$\exists S.Doc, S.Date, S.Time \ (Surgey(S.Doc, S.Date, P.Code, 'S1', S.Time) \wedge S.Date < H.End))))\}$

3) $\{ Doc.Name, Doc.Surname \mid \exists Doc.Code, Doc.Dep, Doc.Hosp \ (Doctor(Doc.Code, Doc.Name, Doc.Surname, Doc.Dep, Doc.Hosp) \wedge$

$\exists S_1.Date, S_1.Pat, S_1.Time \ (Surgey(Doc.Code, S_1.Date, S_1.Pat, 'S2', S_1.Time) \wedge$

$\exists S_2.Date, S_2.Pat, S_2.Time \ (Surgey(Doc.Code, S_2.Date, S_2.Pat, 'S2', S_2.Time) \wedge$

$\exists P_1.Age, P_1.Reg \ (PATIENT(S_1.Pat, P_1.Age, 'MILAN', P_1.Reg) \wedge$

$\exists P_2.Age, P_2.Reg \ (PATIENT(S_2.Pat, P_2.Age, 'MILAN', P_2.Reg) \wedge S_1.Date = '1/08/2021' \wedge$

$S_2.Date = '1/08/2021' \wedge S_1.Pat \neq S_2.Pat))))\}$

4) $\{ S.Type \mid \forall D\,Doc, D.name, D.Surname, D.Dep, D.Hosp \ (Doctor(S.Doc, D.name, D.Surname, D.Dep, D.Hosp) \rightarrow$

$\exists S.Date, S.Pat, S.Time \ (Surgey(D\,Doc, S.Date, S.Pat, S.Type, S.Time)) \wedge S.Date \geq '1/07/2021' \wedge$

$S.Date \leq '31/07/2021')\}$

1.a.

```
SELECT Doc.name, Doc.Surname
FROM Department Dep UNNEST Dep.Doctors Doc
    JOIN Patient P UNNEST P.Hospitalization H ON H.Department_id = Dep.id
WHERE
        ∧ Doc.id = Dep.Chief_id
        ∧ Dep.id NOT IN ( SELECT Dep.id
                          FROM PATIENT P UNNEST HOSPITALIZATION H1
                          JOIN Departimen Dep ON KEYS H.Department_id
                          WHERE P.town = "Rome" ∧
                          NOT (H1.END < '1/08/2021 ∨ H1.START > '31/08/2021')
```

1.b.

```
SELECT P.Code, P.Age
FROM Patien P UNNEST P.HOSPITALIZATON H UNNEST H.SURGEY S
JOIN DEPARTIMENT Dep ON KEYS H.Department_Id
WHERE P.Region = 'LOMBARDY' ∧ H.START ≥ '1/07/2021' ∧ H.START ≤ '31/07/2021'
    ∧ S.date < H.end ∧ S.Type = 'S1' ∧ Dep.Municipality = 'Verona'
```

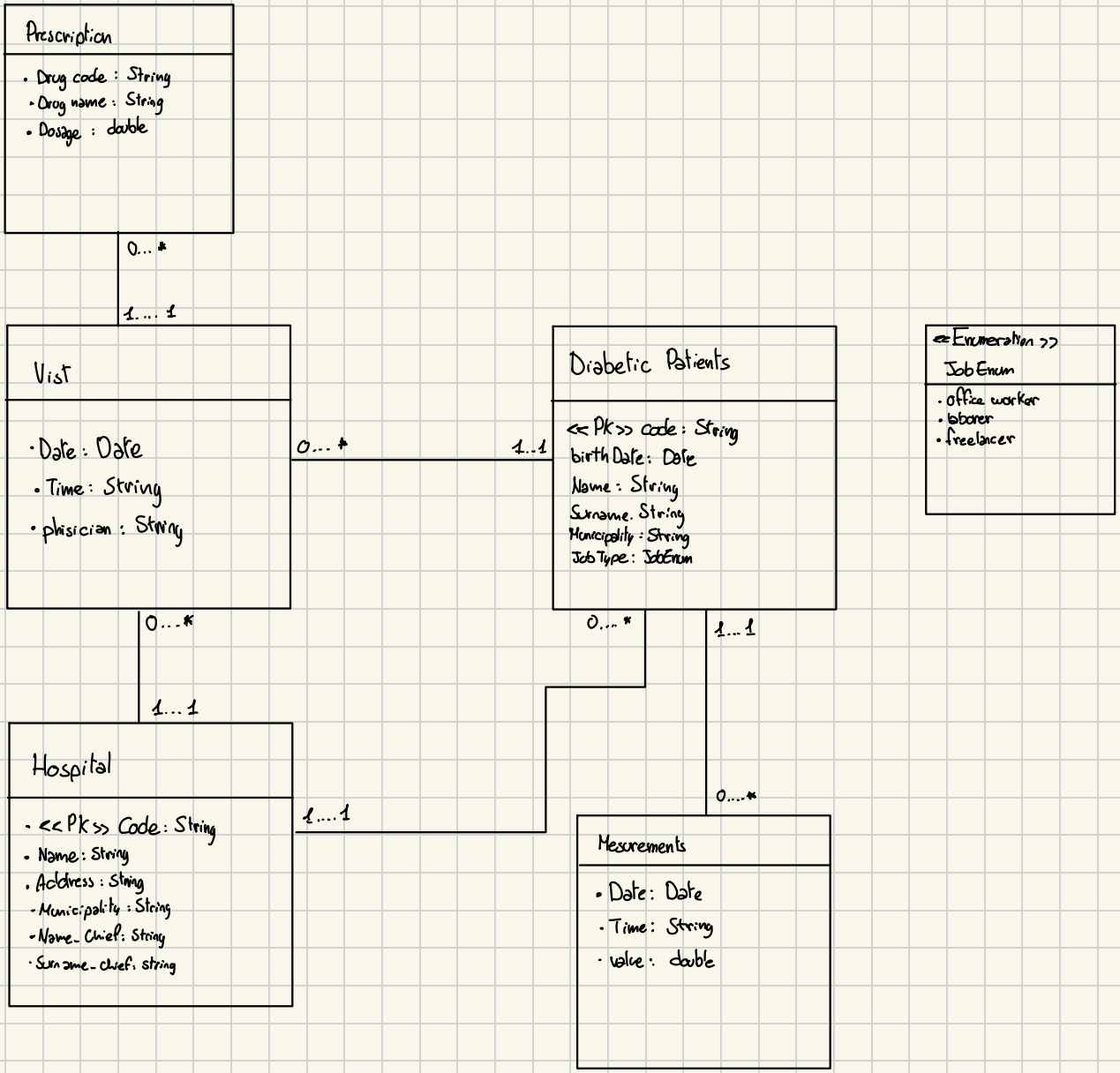
1.c.

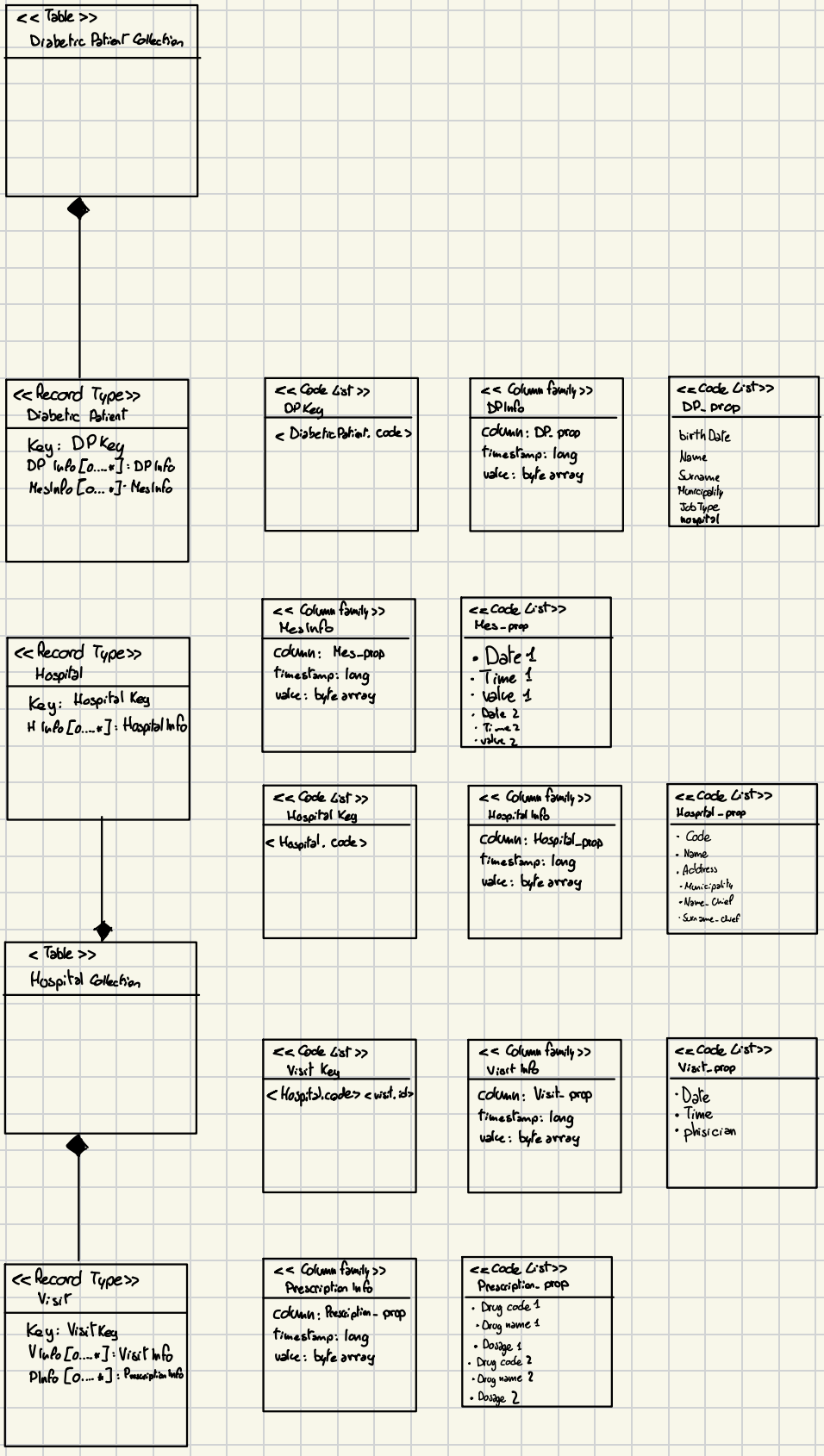
```
SELECT Doc.name, Doc.Surname
FROM DEPARTMENT DEP UNNEST DEP.DOCTORS Doc
JOIN PATIENT P1 UNNEST P1.HOSPITALIZATION H1 UNNEST H1.SURGEY S1 ON DOC.id = S1.Doctor_id
JOIN PATIENT P2 UNNEST P2.HOSPITALIZATION H2 UNNEST H2.SURGEY S2 ON Doc.id = S2.Doctor_id
WHERE P1.Town = 'MILAN' ∧ P2.Town = 'MILAN' ∧ S1.DATE = '1/8/2021' ∧ S2.DATE = '1/8/2021'
    ∧ S1.TYPE = 'S2' ∧ S2.TYPE = 'S2'
```

1.d

```
SELECT S.TYPE
FROM PATIENT P UNNEST P.HOSPITALIZATION H UNNEST H.SURGEY S
JOIN DEPARTMENT AS DEP UNNEST DEP.DOCTORS DOC ON KEYS S.DOCTOR_ID
WHERE S.DATE ≥ "1/07/2021" ∧ S.DATE ≤ "31/07/2021"
GROUP BY S.TYPE
HAVING COUNT(DISTINCT DOCTOR_id) = (SELECT COUNT(*) FROM DEP.DOCTORS)
```

2. Given the following requirements, design a database at conceptual level using UML and translate the class diagram into a physical data model for the HBase system. The information system we are considering deals with the management of diabetic patients (DP) followed by the hospitals of Northern Italy.
For each DP the system stores: a unique code, the date of birth, the name, the surname, the municipality of residence, the job type and the hospital that follows the patient. The job type is a domain of fixed values: office worker, laborer, freelancer, etc. Moreover, the historical sequence of all her/his measurements of glycaemia is stored by means of a list of triples of values: (date, time, value).
For each visit of the patient at the hospital the system stores: the patient, the date, the time, the physician and the prescriptions of drugs. For each prescription: the drug code, the drug name and the dosage are stored. For each hospital the system stores the following attributes: unique code, name, address, and municipality where it is located, together with the name and surname of the doctor that is the chief of the hospital.




<< Table >>
Diabetic Patient Collection
<<Record Type>>
Diabetic Patient
Key: DP Key
DP Info [0....*]: DP Info
Mes Info [0....*]: Mes Info
<< Code List >>
DP Key
< Diabetic Patient. code >
<< Column family >>
DP Info
column: DP_prop
timestamp: long
value: byte array
<< Code List >>
DP_prop
birth Date
Name
Surname
Municipality
Job Type
hospital
<< Column family >>
Mes Info
column: Mes_prop
timestamp: long
value: byte array
<< Code List >>
Mes_prop
• Date 1
• Time 1
• value 1
• Date 2
• Time 2
• value 2
<<Record Type>>
Hospital
Key: Hospital Key
H Info [0....*]: Hospital Info
<< Code List >>
Hospital Key
< Hospital. code >
<< Column family >>
Hospital Info
column: Hospital_prop
timestamp: long
value: byte array
<< Code List >>
Hospital_prop
• Code
• Name
• Address
• Municipality
• Name_Chief
• Surname_chief
< Table >>
Hospital Collection
<< Code List >>
Visit Key
< Hospital).code> < visit. id>
<< Column family >>
Visit Info
column: Visit_prop
timestamp: long
value: byte array
<< Code List >>
Visit_prop
• Date
• Time
• physician
<<Record Type>>
Visit
Key: Visit Key
V Info [0....*]: Visit Info
P Info [0....*]: Prescription Info
<< Column family >>
Prescription Info
column: Prescription_prop
timestamp: long
value: byte array
<< Code List >>
Prescription_prop
• Drug code 1
• Drug name 1
• Dosage 1
• Drug code 2
• Drug name 2
• Dosage 2

```
SELECT E1.Code, E2.Code, E1.D
FROM EPIDEMIC E1, EPIDEMIC E2, EPIDEMIC-EVOLUTION EV1, EPIDEMIC-EVOLUTION EV2, MUNICIPALITIES M
WHERE E1.START_VT ≥ '1/08/2021' ∧ E1.START_VT ≤ '30/06/2021' ∧ E2.START_VT < E1.END_VT ∧ E1.Code = EV1.CodeEpidemic ∧
      (EV1.START_VT, EV1.END_VT) OVERLAPS (E2.START_VT, E2.START_VT) ∧ E2.START_VT = EV2.START_VT ∧
      NOT ST_DISJOINT (EV1.EXTENSION, EV2.EXTENSION) ∧
      ∧ NOT EXIST ( SELECT 1
                    FROM EPIDEDEMIC_EVOLUTION EV3
                    WHERE EV3.CODE = E1.CODE ∧ NOT ST_DISJOINT ( EV3.ESTENSION, M.EXTENSION) ∧ M.Name = 'Verona' )
```

E2.Code = EV2.CodeEpidemic

```
SELECT M.CODE, M.NAME, COUNT(*)
FROM Municipalities M, Epidemic E
WHERE E.start_VT ≥ '1/1/2021' ∧ E.START_VT ≤ '31/1/2021' ∧ M.Inhabitants < 1500 ∧
      ST_CONTAINS (M.EXTENSION, E.PointFirstCode)
GROUP BY M.CODE, M.NAME
```

1.a $\{ S.name, S.surname \mid \exists S.code, S.ResTown, S.region \; \exists (student(S.Code, S.Name, S.surname, S.ResTown, S.Region)$

$\wedge \neg \exists D.date, D.Exam, D.RegDate, D.Grade (CALL\_ENR(D.date, D.Exam, S.Code, D.RegDate, D.Grade)$

$\wedge S.Region \neq Veneto \wedge D.date \geq '1/06/2023' \wedge D.Date \leq '31/06/2023' \wedge D.Exam = 'Databases'))\}$

1.b $\{ D.Name, D.Address \mid \exists D.DepChief (Department(D.name, D.Address, D.DepChief) \wedge$

$\exists P.name, P.Surname, P.Nat, P.Dep (Professor(D.DepChief, P.name, P.Surname, P.Nat, P.Dep) \wedge P.nat \neq 'Italy'$

$\wedge \exists C_1.Date, C_1.Time, C_1.Dep, C_1.Room (CALL(C_1.Date, "Software engineering", D.DepChief, C_1.Time, D.name, C_1.Room)$

$\wedge \exists C_2.Date, C_2.Time, C_2.Dep, C_2.Room (CALL(C_2.Date, "Databases", D.DepChief, C_2.Time, D.name, C_2.Room)$

$\wedge C_1.Date = C_2.Date))))\}$

1.c $\{ P.scr, P.nat \mid \exists P.id, P.name, P.dep (Professor(P.id, P.name, P.surname, P.nat, P.dep)$

$\wedge \neg \exists C.Date, C.Exam, C.Time, C.Dep, C.Room (CALL(C.Date, C.Exam, P.id, C.Time, C.Dep, C.Room)$

$\wedge C.Date \geq '1/06/2023' \wedge C.Date \leq '30/06/2023' \wedge C.Room = 'A'))\}$

1.d $\{ S.name, S.surname, S.ResTown \mid \exists S.Code, S.Reg (Student(S.Code, S.name, S.surname, S.Restown, S.Reg) \wedge$

$S.ResTown \neq 'Verona' \wedge$

$\neg \exists C.Date, C.Exam, C.RegDate, C.Grade (Call\_enr(C.Date, C.Exam, S.code, C.Regdate, C.Grade) \wedge C.Grade \leq 28$

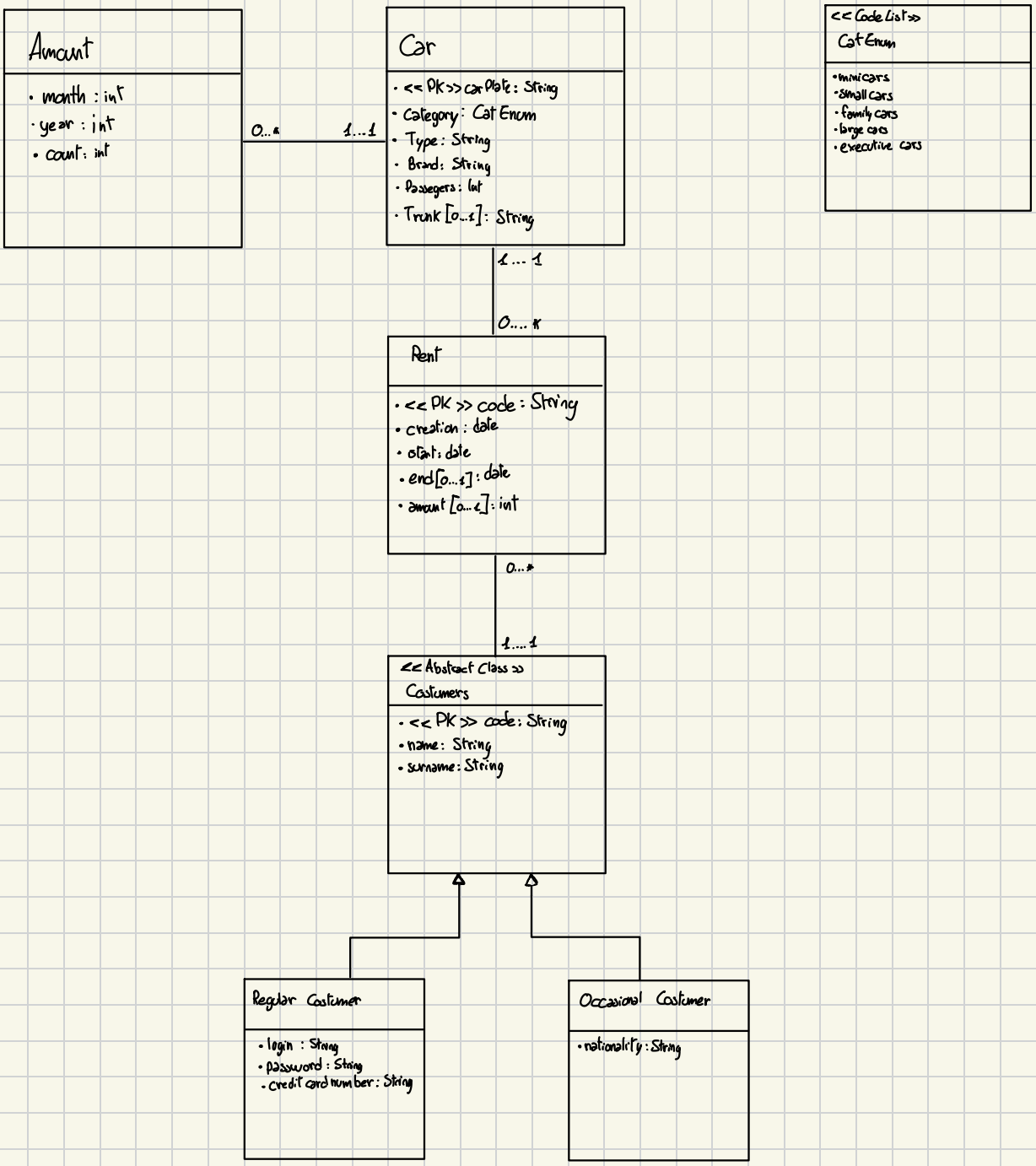
SQL ++

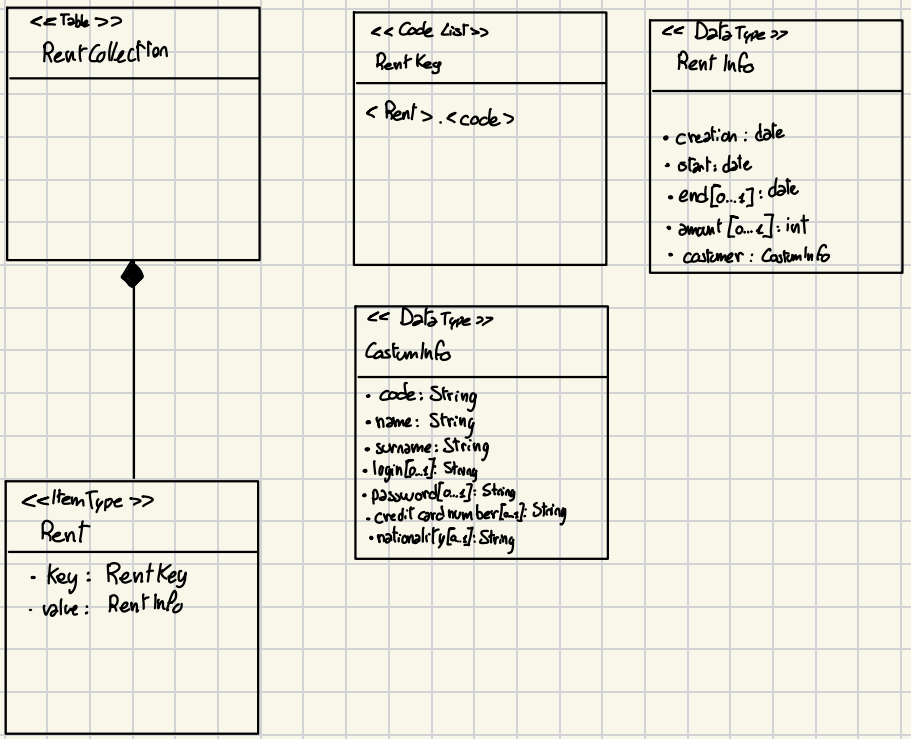
1.a.
```
SELECT S.name, S.surname
FROM Student
WHERE S.Region ≠ 'Veneto' ∧ S.CODE NOT IN ( SELECT S.Code
                                            FROM STUDENT UNNEST EXAM E
                                            JOIN DEPARTMENT D UNNEST D.CALLS C ON C.DATE = E.DATE ∧ C.EXAM = D.EXAM
                                            WHERE C.exam = 'Databases' ∧ C.Date ≥ '1/06/2023' ∧ C.Date ≤ '30/06/2023')
```

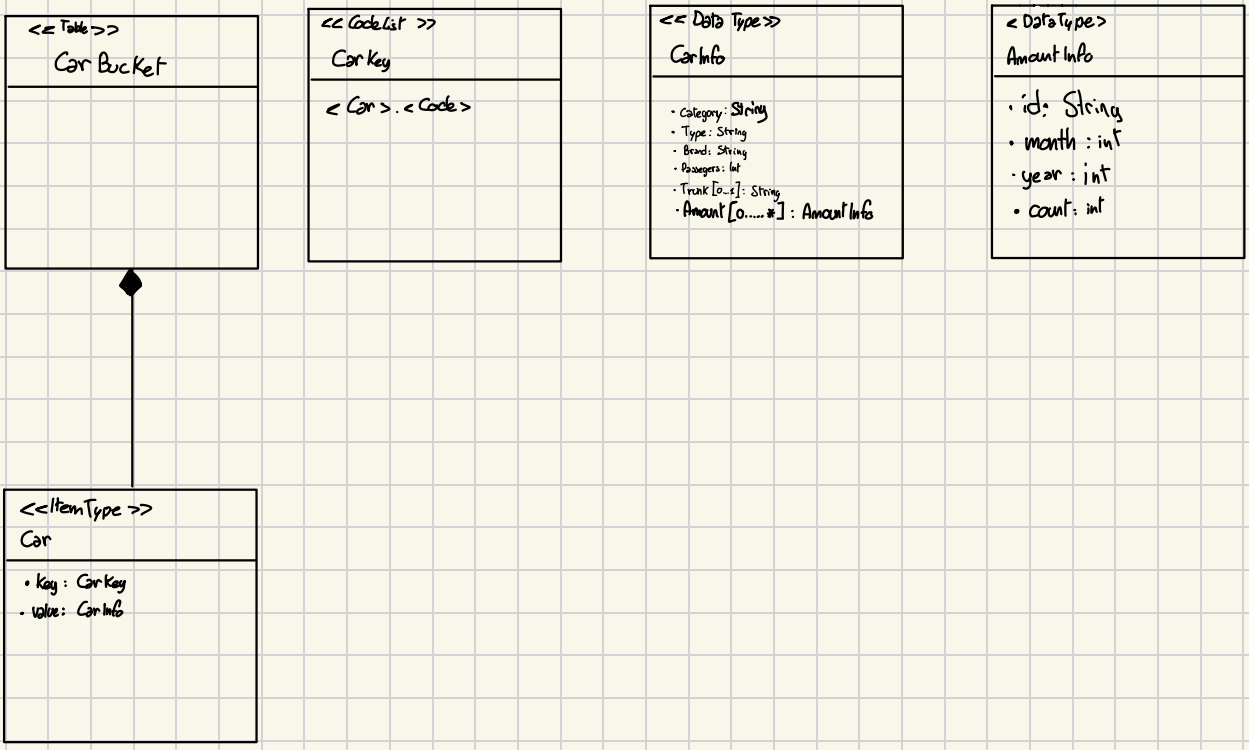
1.b
```
SELECT D.name, D.address
FROM Department D UNNEST D.CALL C
JOIN Department D1 UNNEST D1.CALL C2 ON C2.PROFESSOR.ID = C1.PROFESSOR.ID
JOIN Department D2 UNNEST D2.Profs P2 ON P2.id = C1.Professors id
WHERE D.Dept Chiefs = C1.Professor.id ∧ C1.Exam = 'Databases' ∧ C2.Exam = 'Software Engeneering'
      ∧ P2.NAT ≠ 'Italy'
```

```
SELECT D.name, D.address
FROM Department D UNNEST D.professor P unnest d.call c unnest d.call C2
WHERE D.Dept Chiefs = C.Professor.id
      ∧ C1.Exam = 'Databases' ∧ C2.Exam = 'Software Engeneering'
      ∧ P.NAT ≠ 'Italy' ∧ C1.Date. = C2.Date
      ∧ D.Dept Chiefs = C1.Professor.id
```

UML


Amount
• month : int
• year : int
• count : int
0...*
1...1
Car
• << PK >> carPlate : String
• Category : CatEnum
• Type : String
• Brand : String
• Passegers : int
• Trunk [0...1] : String
<< CodeList >>
CatEnum
• minicars
• small cars
• family cars
• large cars
• executive cars
1 ... 1
0.... *
Rent
• << PK >> code : String
• creation : date
• start : date
• end [0...1] : date
• amount [0...1] : int
0...*
1.... 1
<< Abstract Class >>
Customers
• << PK >> code : String
• name : String
• surname : String
Regular Customer
• login : String
• password : String
• credit card number : String
Occasional Customer
• nationality : String


<<Table>>
RentCollection
<<Code List>>
RentKey
<Rent>.<code>
<< Data Type >>
Rent Info
• creation : date
• start: date
• end[0...1]: date
• amount [0...*]: int
• customer : CustomInfo
<< Data Type >>
CustomInfo
• code: String
• name: String
• surname: String
• login[0..1]: String
• password[0..1]: String
• credit card number[0..1]: String
• nationality[0..1]: String
<<ItemType >>
Rent
• Key : RentKey
• value : Rent Info


<<Table>>
Car Bucket
<<CodeList >>
CarKey
<Car>.<Code>
<< Data Type >>
CarInfo
• Category: String
• Type: String
• Brand: String
• Passengers: Int
• Trunk [0..1]: String
• Amount [0.....*] : AmountInfo
<DataType>
AmountInfo
• id: String
• month : int
• year : int
• count: int
<<ItemType >>
Car
• key : CarKey
• value: CarInfo

3.a

```
SELECT E1.CODE, E2.CODE, E1.DISEASE, E2.DISEASE , ST_ASTEXT(E1.PointFinalCase), ST_ASTEXT(E1.PointFinalCase)
FROM EPIDEMIC E1, EPIDEMIC E2, EPIDEMIC_EVOLUTION EV1, EPIDEMIC_EVOLUTION EV2, PROVINCE P
WHERE E1.START_VT ≥ '1/06/2022' ∧ E1.START_VT ≤ '30/06/2022' ∧
      E2.START_VT ≥ '1/07/2022' ∧ E2.START_VT ≤ '31/07/2022'
      ∧ E1.DISEASE ≠ E2.DISEASE ∧ E1.CODE = EV1.CODEEPIDEMIC ∧ E2.CODE = EV2.CODEEPIDEMIC
      ∧ (EV1.START_VT, EV1.END_VT+1) OVERLAPS (E2.START_VT, E2.START_VT)
      ∧ EV2.START_VT = E2.START_VT ∧ NOT (ST_DISJOINT(EV1.EXTENSION, EV2.EXTENSION)
      ∧ EXIST ( SELECT 1
                FROM EPIDEMIC_EVOLUTION EV_3
                WHERE P.NAME = 'VENICE' ∧ ST.CONTAINS(P.EXTENSION, EV_3.EXTENSION)
```

3b.

```
SELECT P.Region, SUM(P.Inhabitants), COUNT(P.Code)
FROM Province P, Epidemic E1 , Epidemic_evolution EV_1
WHERE E1.Disease = 'D1' ∧ EV1.CodeEpidemic = E1.code ∧ NOT(ST_DISJOINT(EV1.EXTENSION, EV2.EXTESION)
GROUP BY P.Region
```

FFV

```
SELECT E1.CODE, E2.CODE, E1.DISEASE, ST_ASTEXT(E1.PointFirstCase), ST_ASTEXT(E2.PointFirstCase)
FROM EPIDEMIC E1, EPIDEMIC E2, Municipalities M
WHERE E1.START_VT < E2.START_VT ∧ E1.END_VT ≥ '1/05/2021' ∧ E1.END ≤ '31/05/2021' ∧
      E2.END_VT ≥ '1/05/2021' ∧ E2.END ≤ '31/05/2021' ∧
      E1.DISEASE = E2.DISEASE ∧
      EXIST (SELECT 1
             FROM EPIDEMIC_EVOLUTION EV1, EPIDEMIC_EVOLUTION EV2
             WHERE EV1.CODEEPIDEMIC = E1.CODE ∧ EV2.CODEEPIDEMIC = E2.CODE
                   ∧ (EV1.START, EV1.END+1) OVERLAPS ('1/04/2021', '30/04/2021')
                   ∧ (EV2.START, EV2.END+1) OVERLAPS ('1/04/2021', '30/04/2021')
                   ∧ (EV1.START, EV1.END+1) OVERLAPS (EV2.START, EV2.END)
                   ∧ NOT ST_DISJOINT(EV1.EXTENSION, EV2.EXTENSION))
      ∧ NOT EXIST (SELECT 1
                   FROM EPIDEMIC_EVOLUTION EV1
                   WHERE EV1.CODEEPIDEMIC = E1.CODE ∧ NOT ST_DISJOINT(EV1.EXST, M.EXTENSION))
```

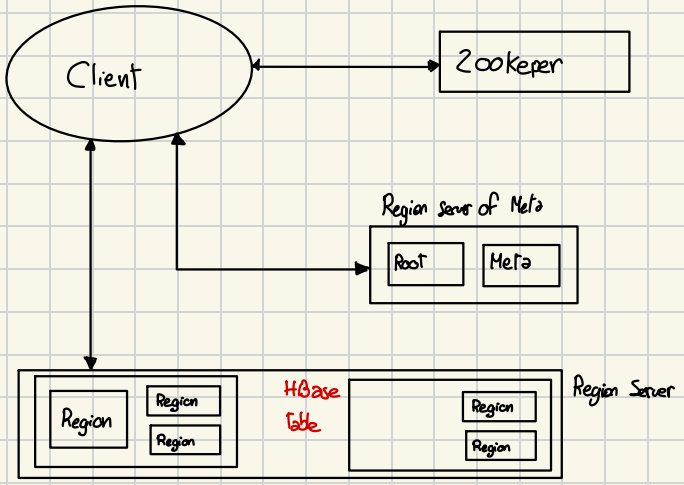
```
SELECT E1.CODE, E1.DISEASE, COUNT(DISTINCT M.CODE)
FROM EPIDEMIC E1, EPIDEMIC_EVOLUTION EV1, MUNICIPALITIES M
WHERE EV1.CODE EPIDEMIC = E1.CODE ∧ E1.START_VT ≥ '1/01/2021' ∧ E1.START_VT ≤ '31/12/2021' ∧
      ∧ NOT ST_DISJOINT(EV1.EXTENSION, M.EXTENSION)
GROUP BY E1.CODE, E1.DISEASE
```

1. Tempo di validità: è il tempo in cui un determinato dato è vero all'interno del contesto dell'applicazione che stiamo modellando

Tempo di transizione: è il tempo in cui un determinato dato è conosciuto dal database (ovvero dal momento di creazione del dato al momento di eliminazione)

Paz	Dep	Hosp	Valid time	Transaction Time
V2472	E1234	H1235	[1/1/2022 12.00 2/2/2022 12.00]	[2/1/2022 12:00 ∞]
V5310	E1352	H5327	[1/2/2022 14.00 3/2/2022 14.00]	[2/1/2022 12:00 ∞]
V7234	E2423	H4321	[3/2/2022 11.00 5/2/2022 14.00]	[3/2/2022 12:00 ∞]

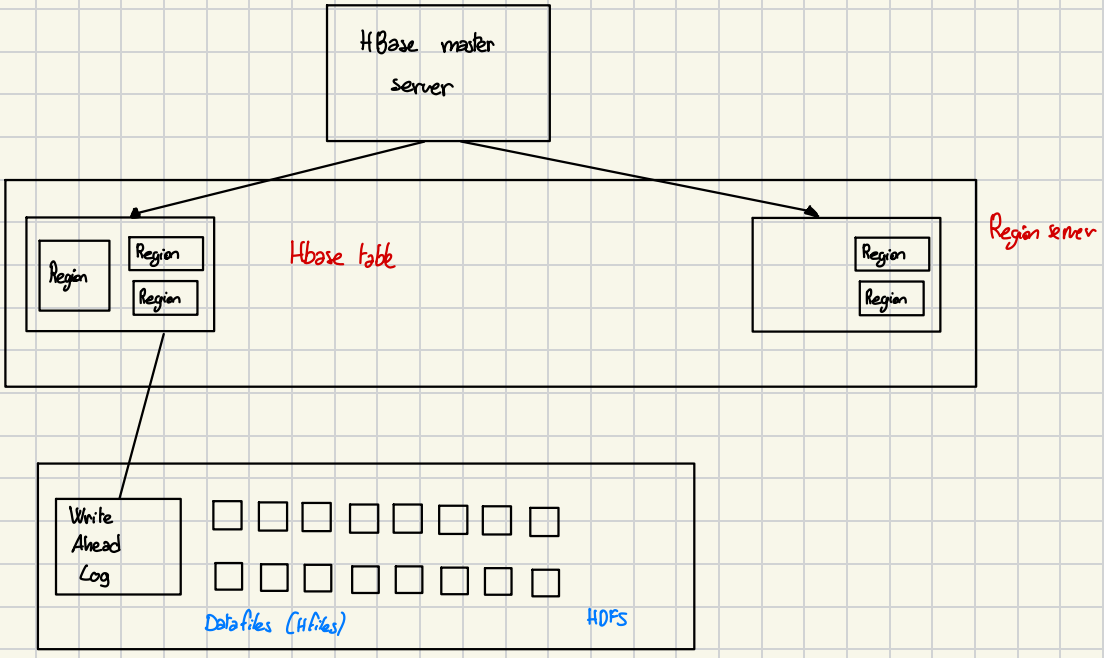
2.



1 Client chiede allo zookeper la locazione del region server che controlla il catalogo delle Hbase Table

2 Il catalogo HBase è interrogato per determinare il region server che gestisce il record che il cliente vuole leggere o scrivere

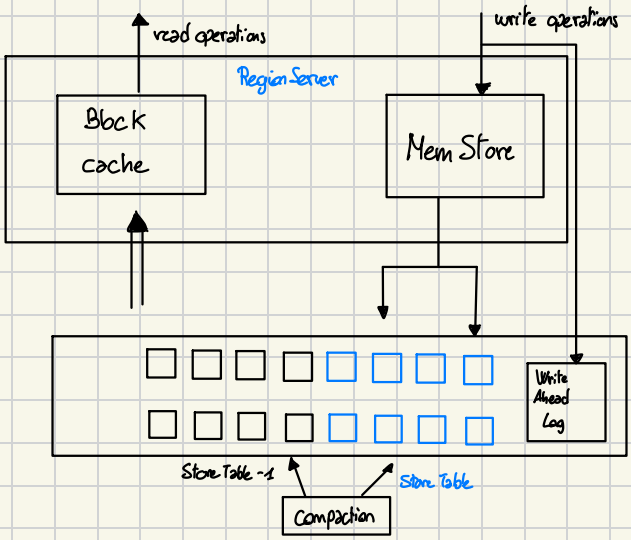
3 Il client manda la richiesta al region server identificato per leggere o scrivere il record



4. Il server master HBase è responsabile del bilanciamento delle regioni fra i region server

5. Per assicurare la durabilità il region server scrive su un file di log chiamato WAL

6. Il flush delle scritture su disco è fatto in maniera asincrona



7 Usando il blocco cache molti record sono copiati in molte memorie

8 Le operazioni di write sono performate nel memstore e viene fatto il flush sul disco successivamente con un purge del log

9 Periodicamente il sistema compatterà store multipli in singoli file. Vengono compattate le righe frammentate su store multipli mentre le righe cancellate vengono rimosse

(3.) λ_1 Overlaps λ_2 quando: $dim(\lambda_1^0 \cap \lambda_2^0) = dim(\lambda_1) = dim(\lambda_2) \wedge \lambda_1 \cap \lambda_2 \neq \lambda_1 \wedge \lambda_1 \cap \lambda_2 \neq \lambda_2$

(4.) L'approccio sharding in MongoDB è implementato da un distinto server mongoDB, un altro server MongoDB detto config server contiene i metadati che possono essere usati per determinare lo shard in cui verranno memorizzati i dati.
Un processo routing permette l'indirizzamento allo shard corretto
Si sceglie una shard key composta da uno o più attributi indicizzati che verranno usati per distribuire i documents tra le shard

La distribuzione dei documents tra le shard è:

- range based: ad ogni shard è assegnato un range di shard-key value
- Hash based: i document sono distribuiti sulla base di una funzione hash tra le varie shard

Durante la vita di un sistema il carico di lavoro può variare e la policy di sharding iniziale può non essere più valida causando sbilanciamenti
MongoDB periodicamente valuta ribilanciamenti ed eventualmente li effettua con i shard chunk l'unità di ribilanciamento che ha dimensione 64 MB. Ciascun shard chunk contiene valori contigui di shard key
- Si possono muovere chunk da uno shard all'altro per ribilanciare

⑤ Consistenza di Cassandra: adotta la tunable consistency, permette di impostare tre valori

- numero di copie da mantenere
- numero di scritture da effettuare prima che la write sia completata
- numero di letture prima che la read sia completata

Write consistency level

- ALL
- Local_All
- Quorum each
- Quorum local
- Any
- One / Two / Three

Read Level Consistency

- ALL
- One / Two / Three
- Quorum Each
- Quorum local
- Local One

read / write	one	quorum	all
one	performance migliori possibile consistenza		letture lente e consistenti: scritture veloci ma non consistenti
quorum		buone prestazioni e buona consistenza	
all	scritture lente e consistenti: letture veloci ma poco consistenti		performance peggiori migliore consistenza

- A meno che la write non sia ad ALL avremo della inconsistenza nel sistema cassandra

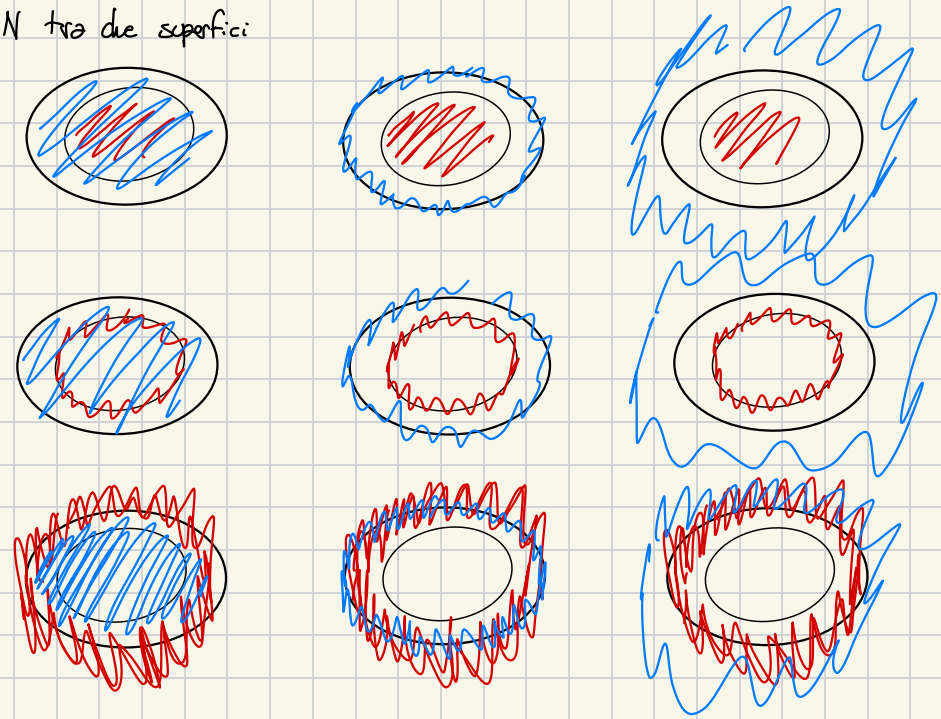
Vengono adottati i sistemi hinted handoff e read repair

- Hinted handoff: consiste di memorizzare a un nodo N un aggiornamento per un nodo No se questo non è disponibile per ricevere tale aggiornamento. Se dopo questo quantitativo di tempo il nodo ricevente non torna disponibile l'aggiornamento andrà perso
- Read repair: processo che avviene in riparazione a un fallimento di hinted handoff, quando una read diretta a più nodi da uno prelevo i dati veri e propri dagli altri leggo i digest hash se questi non combaciano ritorno i dati più recenti e aggiorno gli altri

6. Matrice di Egenhofer

$$\begin{pmatrix} \lambda_1^{\circ} \cap \lambda_2^{\circ} & \lambda_1^{\circ} \cap \delta\lambda_2 & \lambda_1^{\circ} \cap \lambda_2^{-} \\ \delta\lambda_1 \cap \lambda_2^{\circ} & \delta\lambda_1 \cap \delta\lambda_2 & \delta\lambda_1 \cap \lambda_2^{-} \\ \lambda_1^{-} \cap \lambda_2^{\circ} & \lambda_1^{-} \cap \delta\lambda_2 & \lambda_1^{-} \cap \lambda_2^{-} \end{pmatrix}$$

IN tra due superfici



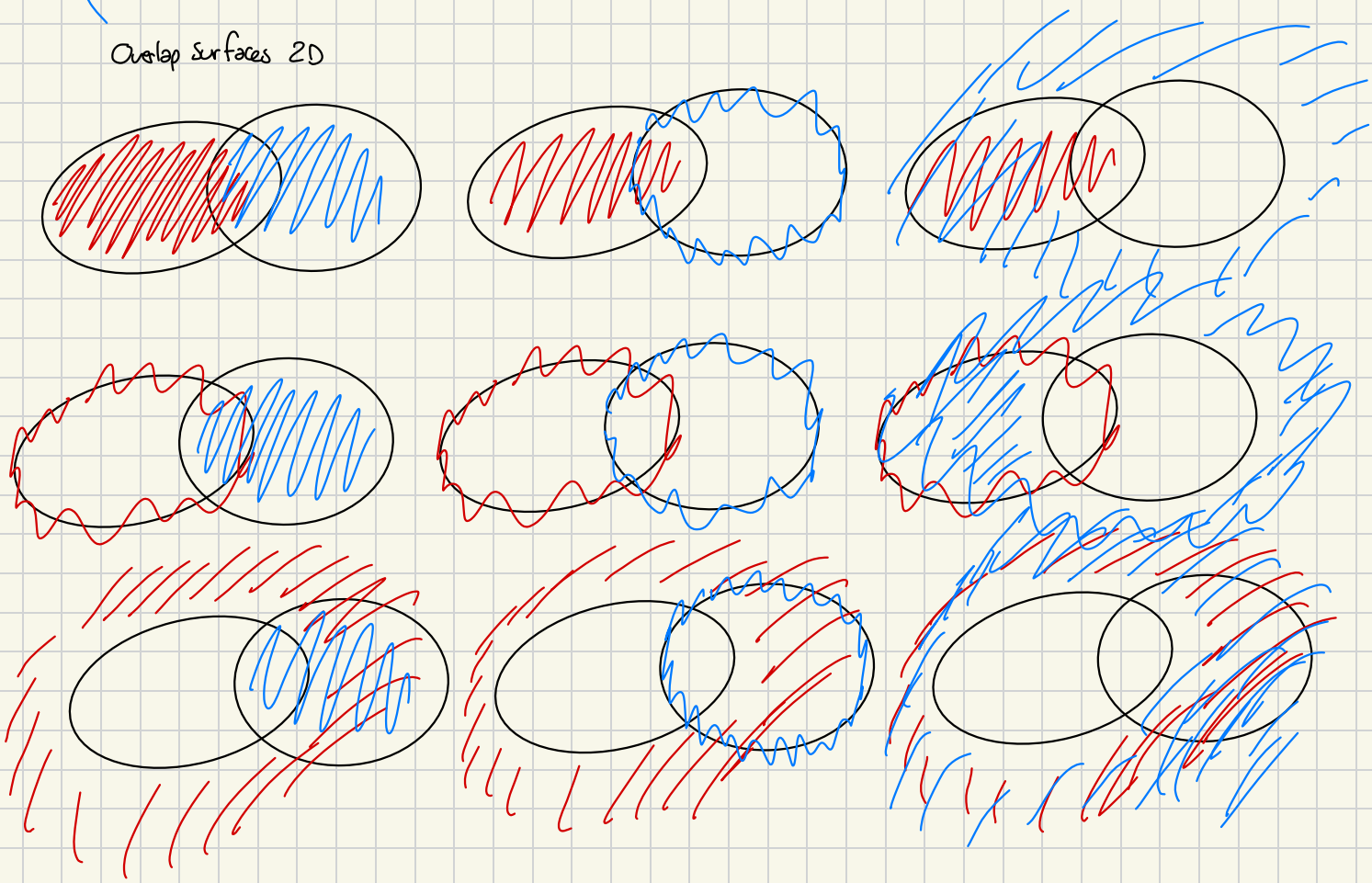
[V F F V f f V V V]

7.

OVERLAP

$$\begin{pmatrix} \lambda_1^{\circ} \cap \lambda_2^{\circ} & \lambda_1^{\circ} \cap \delta\lambda_2 & \lambda_1^{\circ} \cap \lambda_2^{-} \\ \delta\lambda_1 \cap \lambda_2^{\circ} & \delta\lambda_1 \cap \delta\lambda_2 & \delta\lambda_1 \cap \lambda_2^{-} \\ \lambda_1^{-} \cap \lambda_2^{\circ} & \lambda_1^{-} \cap \delta\lambda_2 & \lambda_1^{-} \cap \lambda_2^{-} \end{pmatrix}$$

Overlap surfaces 2D



$$\begin{bmatrix} v & v & v \\ v & v & v \\ v & v & v \end{bmatrix}$$

1.a

$$\{ Dname, DSurname \mid \exists DCode, DDep, DHospital (Doctor(DCode, DName, DSurname, DDep, DHospital)$$
$$\wedge \neg \exists SDate, Spatient, Stype, Stime (Surgery(Dcode, Sdate, Spatient, Stype, Stime)$$
$$\wedge \exists Psurname, Pname, PresTown, Pregion (Patient(Spatient, Psurname, Pname, PresTown, Pregion)$$
$$\wedge SDate \geq '1/06/2024' \wedge SDate \leq '5/06/2024' \wedge PresTown = 'Venice'))) \}$$

1.b

$$\{ P\,name, P\,surname \mid \exists P\,Code, P\,ResTown, P\,Region (Patient(PCode, PSurname, Pname, PresTown, Pregion) \wedge$$
$$\exists HStart, HDep, Hhosp, HEnd (Hosp(HStart, PCode, HDep, Hhosp, HEnd) \wedge$$
$$\exists SDoc, SDate, SType, STime (Surgery(SDoc, SDate, Pcode, SType, STime) \wedge$$
$$\exists DepDepChief, DepAddr, DepMun (Department(HDep, Hhosp, DepDepChief, DepAddr, DepMun) \wedge$$
$$HStartDate \geq '1/02/2020' \wedge H.StartDate \leq '29/02/2020' \wedge Stype = 'S1' \wedge H.end > S.Date$$
$$\wedge P.Region = 'Lombardy' \wedge DepMun = 'Verona')))) \}$$

1.c

$$\{ DepName, Dephosp, DepMun \mid \exists DepDepChief, Dep\,Addres (Department(Depname, Dephosp, DepDepchief, Depaddr, Dep\,Mun)$$
$$\wedge \exists H_1 Start, H_1 Pat, H_1 End (Hosp(H_1 start, H_1 Pat, Depname, Dephosp, H_1 End)$$
$$\wedge \exists H_2 Start, H_2 Pat, H_2 End (Hosp(H_2 Start, H_2 Pat, Depname, Dephosp, H_2 End))$$
$$\wedge \exists P_1 Sur, P_1 name, P_1 Reg (Patient(H_1 Pat, P_1 Sur, P_1 Name, 'Rame', P_1 Reg)$$
$$\wedge \exists P_2 Sur, P_2 name, P_2 Reg (Patient(H_2 Pat, P_2 Sur, P_2 Name, 'Rame', P_2 Reg)$$
$$\wedge H_1 Start = '1/10/2021' \wedge H_2 Start = '1/10/2021' \wedge H_1 Code \neq H_2 Code)))))\}$$

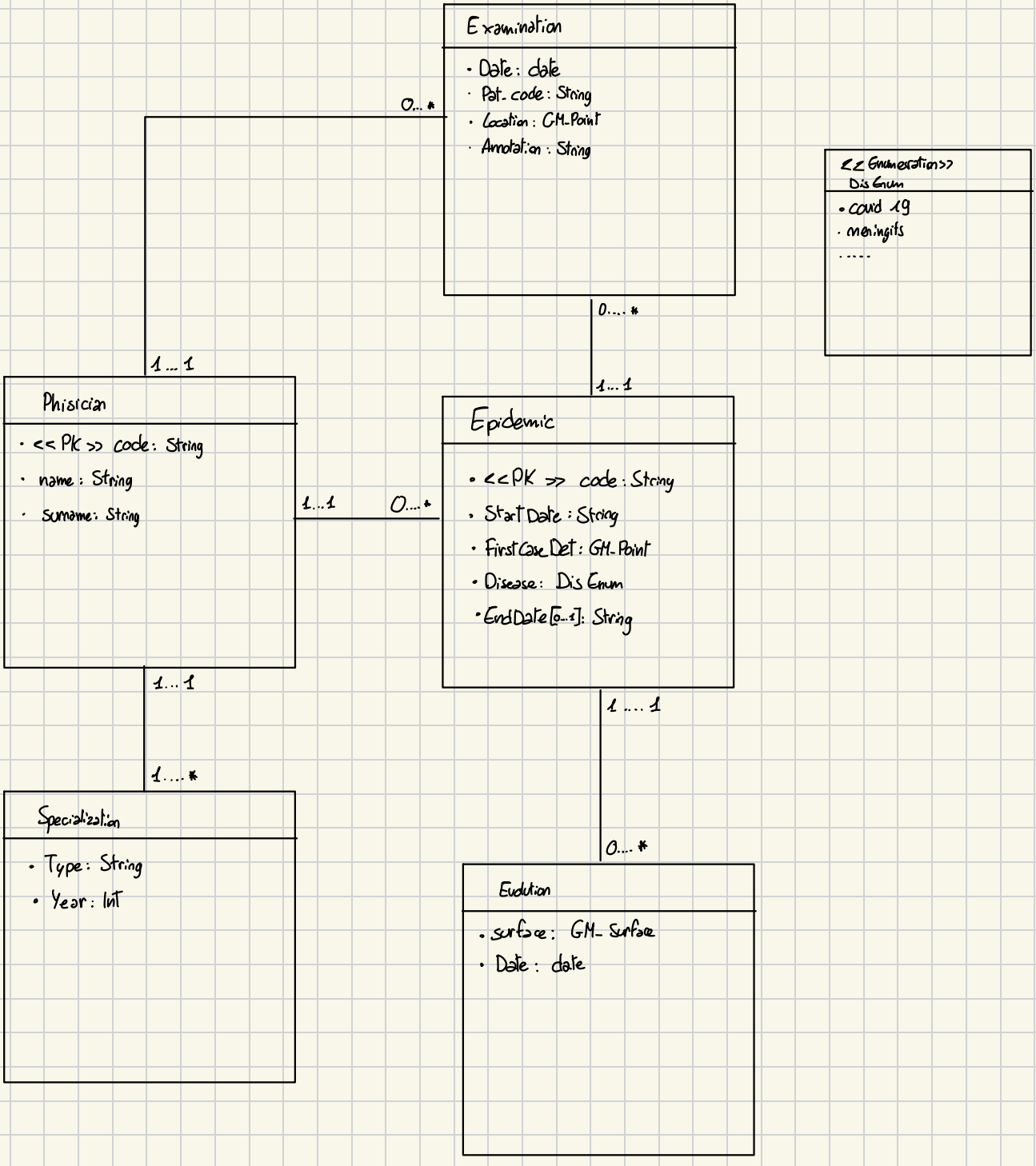
1.d

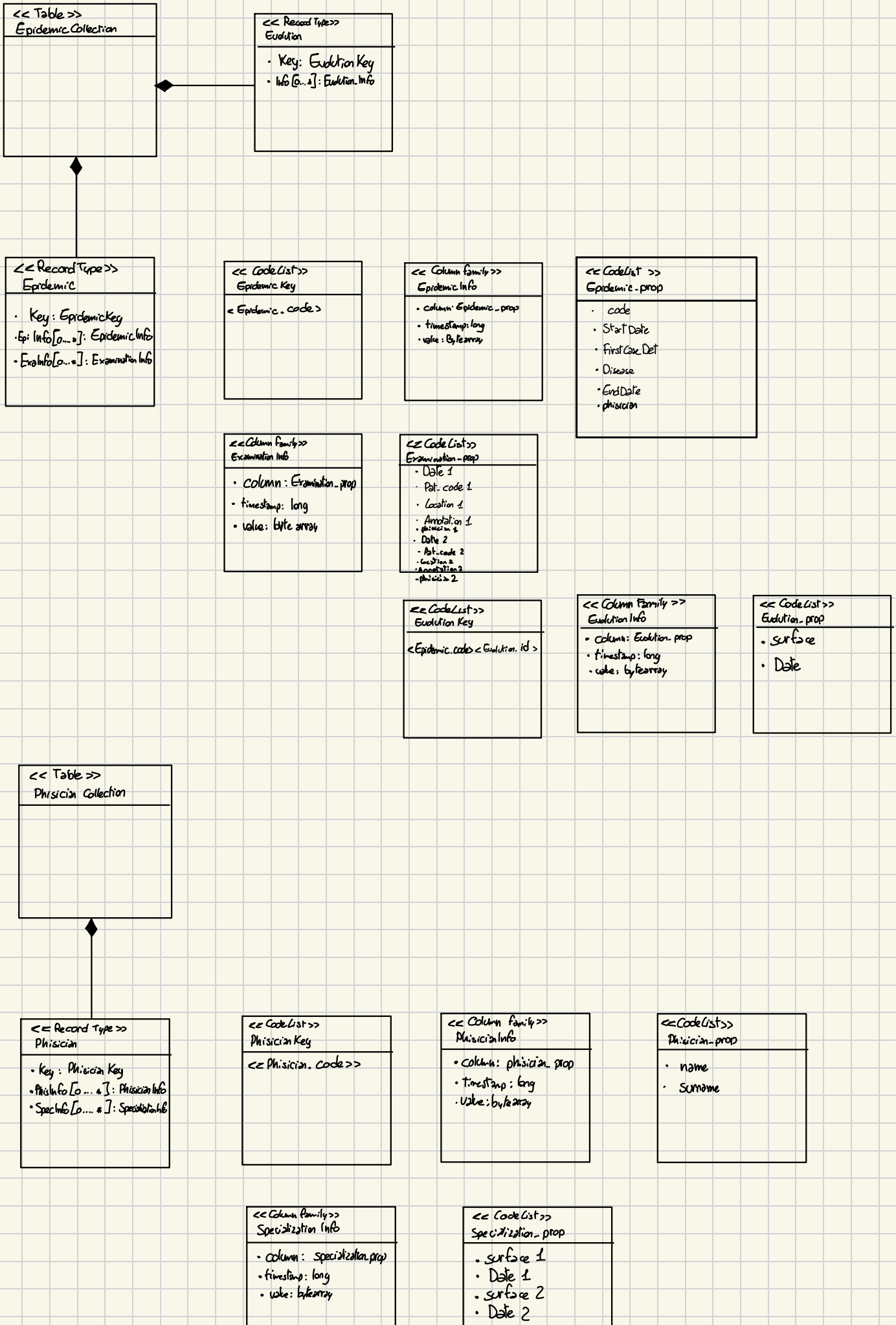
$$\{ Stype \mid \forall DCode, Dname, Dsurname, Ddep, Dhosp (Doctor(Dcode, Dname, Dsurname, Ddep, Dhosp) \rightarrow$$
$$\exists Sdate, SPat, STime (Surgery(Dcode, Sdate, Spat, Stype, Stime) \wedge S.date \geq '1/03/2020'$$
$$\wedge S.date \leq '1/03/2020')) \}$$

1.2

```
SELECT D.name, D.surname
FROM Department Dep unnest Dep.Doctors Doc
WHERE Doc.id not in (select Doc.id
                     from Department Dep unnest Dep.Doctors Doc
                     Join Patient P unnest P.hospitalization * unnest H.surgery S ON S.Doctor_id = Doc.id
                     where S.date >= '1/06/2024' ∧ S.Date ≤ '5/06/2024' ∧ P.ResTown = 'Venezia')
```

2




<< Table >>
Epidemic_Collection
<< Record Type >>
Evolution
• Key: Evolution Key
• Info [0...1]: Evolution_Info
<< Record Type >>
Epidemic
• Key: EpidemicKey
• Epi Info[0...*]: EpidemicInfo
• ExaInfo[0...*]: Examination Info
<< CodeList >>
Epidemic Key
< Epidemic_code >
<< Column family >>
Epidemic Info
• column: Epidemic_prop
• timestamp: long
• value: ByteArray
<< CodeList >>
Epidemic_prop
• code
• Start Date
• First Case Det
• Disease
• End Date
• phisician
<<Column family>>
Examination Info
• column: Examination_prop
• timestamp: long
• value: byte array
<< CodeList >>
Examination_prop
• Date 1
• Pat._code 1
• Location 1
• Annotation 1
• phisician 1
• Date 2
• Pat.code 2
• Location 2
• Annotation 2
• phisician 2
<< CodeList >>
Evolution Key
<Epidemic.code> < Evolution. id >
<< Column Family >>
Evolution Info
• column: Evolution_prop
• timestamp: long
• value: byteArray
<< CodeList >>
Evolution_prop
• surface
• Date
<< Table >>
Phisician Collection
<< Record Type >>
Phisician
• Key: Phisician Key
• PhisInfo [0... *]: PhisicianInfo
• SpecInfo [0.... *]: SpecializationInfo
<< CodeList >>
Phisician Key
<< Phisician. code >>
<< Column family >>
Phisician Info
• column: phisician_prop
• Timestamp: long
• Value: byteArray
<<CodeList>>
Phisician_prop
• name
• surname
<<Column family>>
Specialization Info
• column: Specialization_prop
• timestamp: long
• value: byteArray
<< CodeList >>
Specialization_prop
• surface 1
• Date 1
• surface 2
• Date 2
....

2 $R_1(A, C, D)$
$R_2(A, B)$
$R_3(B, C, D, R)$
$R_4(D, P)$
$R_5(P, S)$ ← può non servire perché contenuto
$R_6(S, T, P, Q)$
$R_7(A, C, T)$

4a.
```
SELECT E1.Code, E2.Code, E1.Disease
FROM Epidemic E1, Epidemic E2, Epidemic_Evolution EV1, Epidemic_Evolution EV2, Municipalities M
WHERE E1.start_VT ≥ '1/1/2020' ∧ E1.start_VT ≤ '31/12/2020' ∧ E2.start_VT < E1.end_VT ∧ EV1.codeEpidemic = E1.code ∧
  ∧ E1.disease = E2.disease ∧ (EV1.Start_VT, EV1.End_VT+1) OVERLAPS (E2.start_VT, E2.start_VT) ∧
    E2.code = EV2.codeEpidemic ∧ EV2.start_VT = E2.start_VT ∧ NOT ST_DISJOINT(EV1.EXTENSION, EV2.EXTENSION) ∧
  EXIST (SELECT 1
         FROM EPIDEMIC_EVOLUTION EV
         WHERE EV.Code epidemic = EV1.CodeEpidemic ∧ M.Name = 'Venice' ∧ NOT ST_DISJOINT(EV1.EXTENSION, M.EXTENSION)
```

4.b.
```
SELECT M.Code, M.name, COUNT(DISTINCT E.CODE)
FROM Municipality M, EPIDEMIC E1
WHERE M.Inhabitants < 2500 ∧ E1.StartVT ≥ '1/01/2021' ∧ E1.ENDVT < '31/12/2021'
    ∧ ST_CONTAINS(M.EXTENSION, E1.Point First Case)
GROUP BY M.CODE, M.name
```

Valid Time: tempo in cui un dato è vero nel contesto che il database analizza
Transaction Time: tempo in cui il database è a conoscenza di un determinato dato, dalla sua creazione alla sua eliminazione

P32	Dep	Hosp	Valid time	Transaction Time
V2472	E1234	H1235	[1/1/2022 13.00 2/2/2022 12.00]	[2/1/2022 12:00 ∞]
V5310	E1352	H4327	[1/2/2022 14.00 3/2/2022 14.00]	[2/1/2022 12:00 ∞]
V7234	E2423	H4321	[3/2/2022 11.00 5/2/2022 16.00]	[3/2/2022 12:00 ∞]

1.a $\{ Pname, Psurname \mid \exists Pcode, Page, Ptown, Pregion (Patient(Pcode, Psurname, Pname, Page, Ptown, Pregion) \wedge$

$\exists H\_StartDate, H\_Department, Hhospital, Hend Date (Hosp(H\_startDate, Pcode, HDepartment, Hhospital, HendDate)$

$\wedge \exists S_1 Doctor, S_1 Date, S_1 Type, S_1 Time (Surgey(S_1 Doctor, S_1 Date, Pcode, S_1 Type, S_1 Time) \wedge$

$\exists S_2 Doctor, S_2 Date, S_2 Type, S_2 Time (Surgey(S_2 Doctor, S_2 Date, Pcode, S_2 Type, S_2 Time)$

$\wedge \exists DepDepChief, DepAdress, DepTown, Dep\#employee (Department(H\_Department, Hhospital, DepDepChief, DepAddress, DepTown)$

$\wedge Page > 50 \wedge DepTown = 'Verona' \wedge Pregion = 'Lombardy' \wedge S_1.Date < 'today' - Interval(5\ days)$

$\wedge S_2.Date < 'today' - Interval(5\ days) \wedge Hosp.StartDate \leq 'today' \wedge Hosp.EndDate = 'NULL'$

$\wedge S_1 Type \neq S_2 Type \}$

→ Assumiamo a null i ricoveri non inseriti

1.b $\{ Dname, Dsurname \mid \exists Dcode, DDep, Dhos (Doctor(Dcode, Dname, Dsurname, DDep, Dhos) \wedge$

$\exists DepDepchief, DepAddress, DepTown, Dep\#employee (Department(DDep, Dhos, DepDepchief, DepAddress, DepTown, Dep\#employee)$

$\wedge \neg \exists SDate, SPatient, Stime (Surgey(Dcode, SDate, Spatient, 'S1', Stime)$

$\wedge \exists Psur, Pname, Page, Ptown, Pregion (Patient(Spatient, Psur, Pname, Page, Ptown, Pregion)$

$\wedge DepTown = 'Padova' \wedge S.Date > '1/06/2021' \wedge S.Date < '30/06/2021' \wedge Page > 80)))) \}$

1.c $\{ Dep1Name, Dep1Hosp, DSurname \mid \exists Dep1DepChief, Dep1Address, Dep1Town, Dep1\#employee$

$(Department(Dep1Name, Dep1Hosp, Dep1DepChief, Dep1Address, Dep1Town \wedge Dep1\#employee) \wedge$

$\exists Dname (Doctor(Dep1DepChief, Dname, Dsurname, Dep1Name, Dep1Hosp) \wedge$

$\neg \exists Dep2Name, Dep2hosp, Dep2DepChief, Dep2Address, Dep2Town, Dep2\#employee$

$(Department(Dep2Name, Dep2Hosp, Dep2DepChief, Dep2Address, Dep2Town \wedge Dep2\#employee) \wedge$

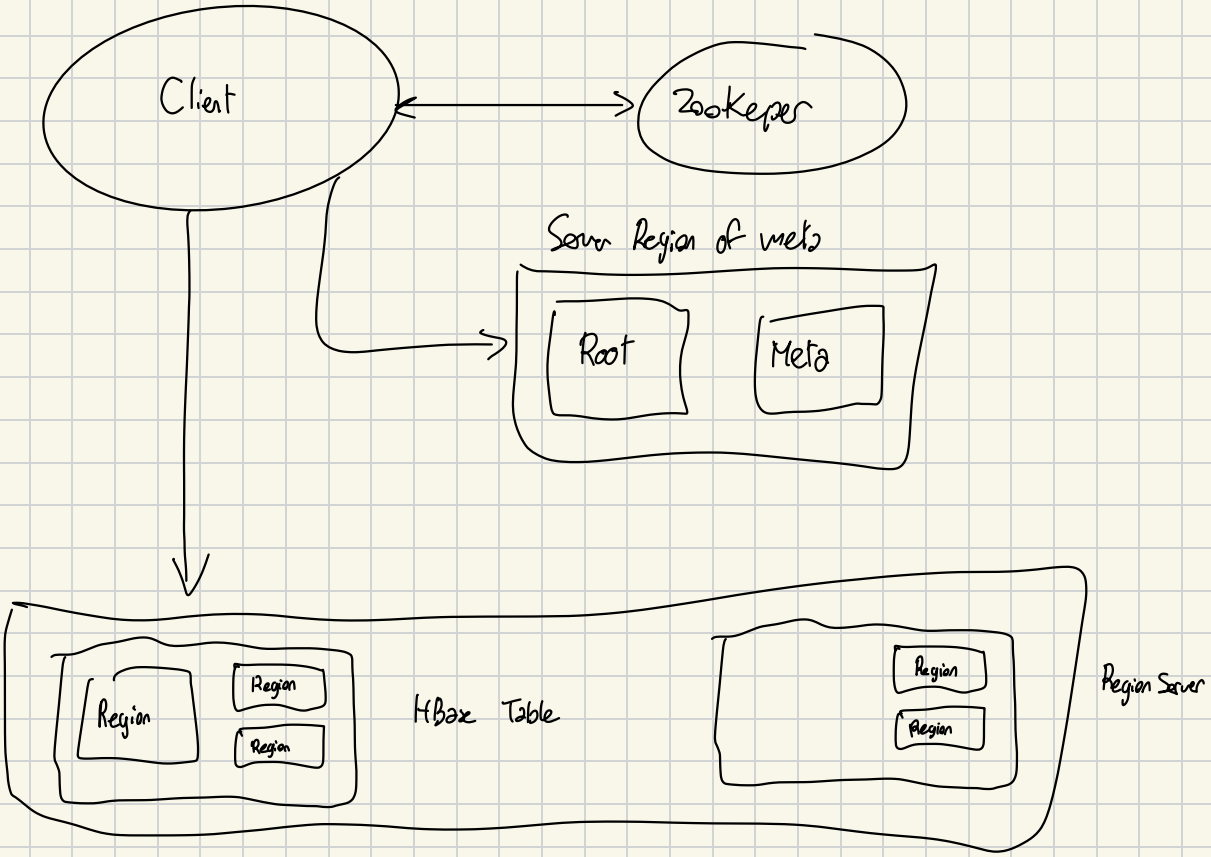
$Dep1\#employee < Dep2\#employee))) \}$

1.d $\{ DepName, DepHosp \mid \exists DepDepChief, DepAddr, DepTown, Depemp$

$(Department(Depname, DepDepChief, DepAddr, DepTown, Depemp)$

$\wedge \forall Day (Calendar(Day) \rightarrow \exists HStart, HPat, HEnd (Hosp(HStart, HPat, Depname, Dephosp, HEnd)$

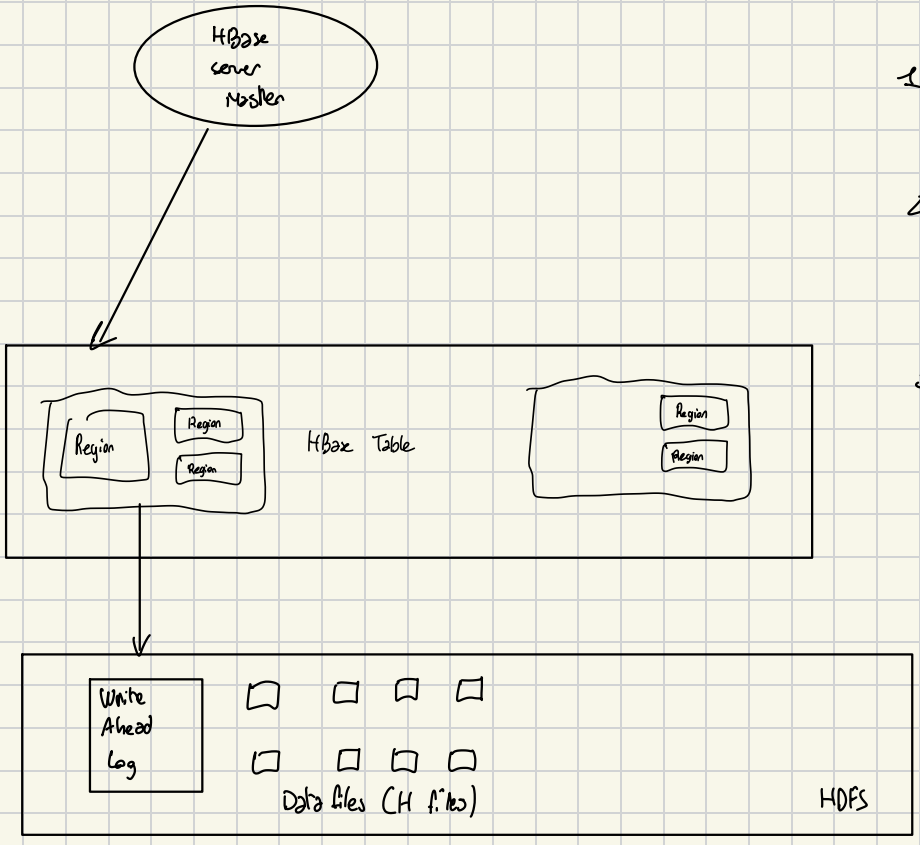
$\wedge H.Start \leq Day \wedge H.End \geq Day \wedge H.Day \geq '1/08/2020' \wedge H.Day' \leq 31/08/2020'))) \}$



1 Il Client interroga lo zookeper per capire dove si trova il region server of Meta che è come un catalogo che ci indica il region server dove trovare dati da leggere e dove andarli a scrivere

2 Il client interroga il catalogo che risponde

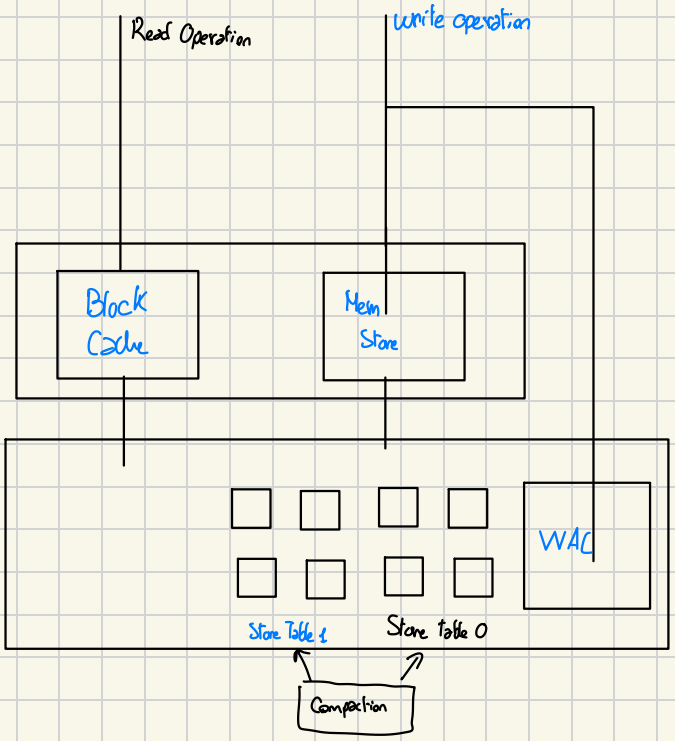
3 Va a scrivere / leggere sul catalogo indicato



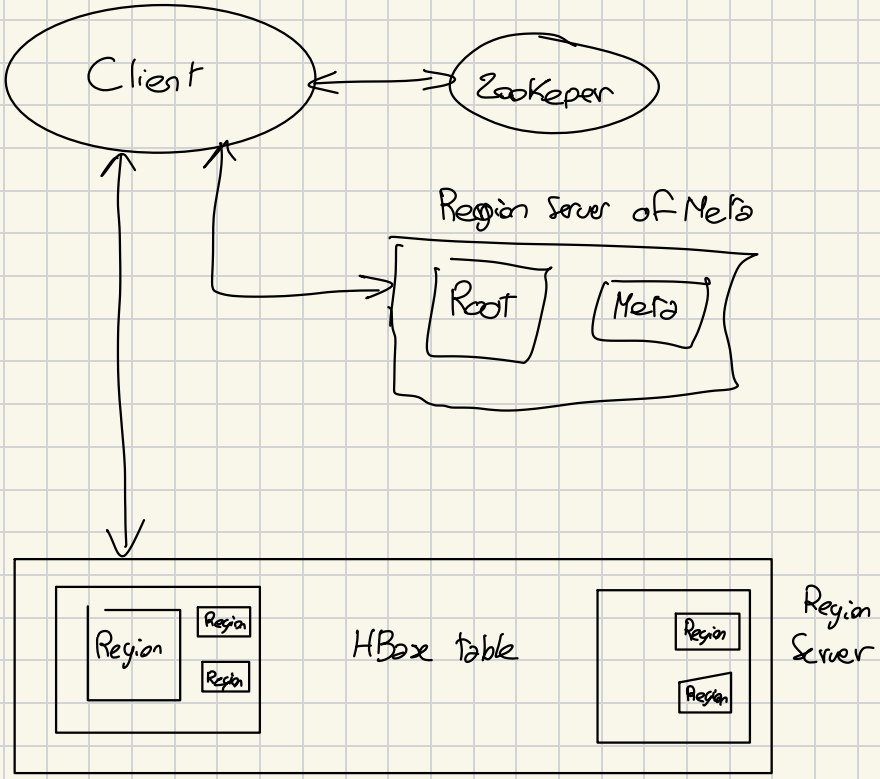
1 L'HBase server master è responsabile del bilanciamento del sistema

2 Per mantenere la durabilità i region server scrivono sul write ahead log

3 Avviene successivamente il flush del log sui file



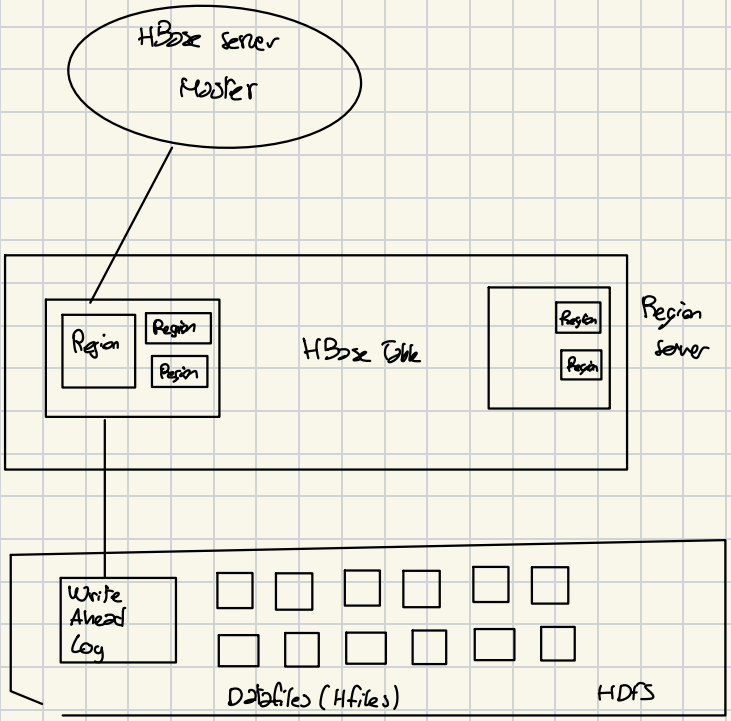
- Con il mem store molti record vengono scritti in molti file
- Le write avengono sul mem store, solo dopo avengono sul disco con un flush del log
- Avviene la compattazione di dati scritti su file multipli e vengono cancellate le righe eliminate



1 Client chiede a/a Zookeper il riferimento al Region server di meta

2 Una volta ottenuto il riferimento si interroga questo che è come un catalogo e ci indicherà il riferimento alla chiave relativo al dato che voglio leggere e/o scrivere

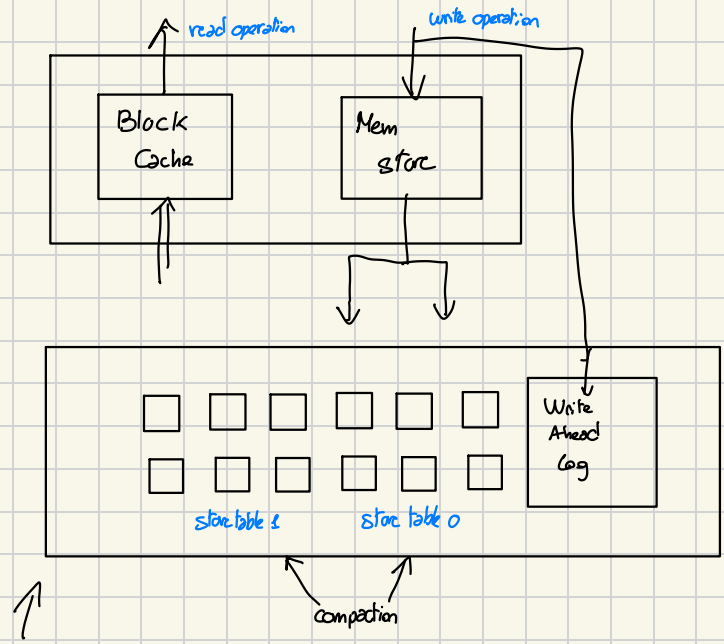
3 Una volta ottenuto il riferimento si scrive o/e legge sul region server che ci è stato indicato



1 L' Hbase master server è responsabile di mantenere il bilanciamento del sistema

2 Per assicurare la durabilità i region server scrivono nel Write Ahead Log

3 Le scritture sui Datafiles avvengono in maniera asincrona con un flush del log



HBase distribute architecture

1 Store multipli tramite il block cache vengono memorizzati internamente

2 L'operazione di write avviene nel memstore e poi verrà propagata ai file mediante una purge dei file di log

3 La compattazione di store multipli in un unico file avviene periodicamente e vengono rimossi store eliminati

MongoDB Shard

Il mongo DB shard è adottato usando un mongoDB server aggiuntivo, uno detto config server usato per mantenere i metadati che ci permetteranno di memorizzare le cose nello shard corretto.
Un processo di routing che ci permetterà di indirizzare i dati nello shard corretto.
Una shard key basata sugli attributi anch'essa per gestire lo sharding e per capire dove verranno memorizzati i dati

2 Approcci:

- Range based: ogni shard ha un range di shard key contigui da gestire
- Hash based: hanno funzione hash che mi indirizza i dati nelle varie shard

La policy di sharding durante la vita del sistema può non essere più valida
Avverrà sbilanciamento → ci posso rimediare modificando la policy di sharding e operando ribilanciamento tramite gli shard chunk che hanno dim 64 Mb e che posso spostare da uno shard a un altro

Cassandra consistency

Per gestire la consistenza in cassandra ci è consentito impostare 3 parametri

- numero di copie da mantenere
- numero di scritture da effettuare per considerare la write operation conclusa
- numero di letture da effettuare per considerare una read operation conclusa

Possibili valori per parametro di write

- All
- One / two / three
- Each_Quorum
- Local_Quorum
- Any

Possibili valori per parametro di read

- All
- One / two / three
- Each_Quorum
- Local_Quorum
- Local_One

write \ read	One	Quorum	All
One	Prestazioni ottime, massima consistenza		write veloce e poco consistente, read consistente ma poco performante
Quorum		buone prestazioni e buona consistenza	
All	write lenta ma consistente, read lenta e poco consistente		consistenza massima, massime prestazioni

Hinted handoff: se un nodo No non è disponibile un altro nodo N può immagazzinare un aggiornamento dei dati per No al posto suo eventualmente se No torna disponibile ricevera l'aggiornamento, se dopo un po' di tempo (3 ore ciò non avviene) l'aggiornamento andrà perso

Read repair: procedura che avviene in seguito a un fallimento di hinted handoff quando si effettua una lettura di un nodo si prende il dato vero e proprio, da altri invece si prende il digest hash se i valori non corrispondono si prende il dato più recente e si aggiornano gli altri nodi