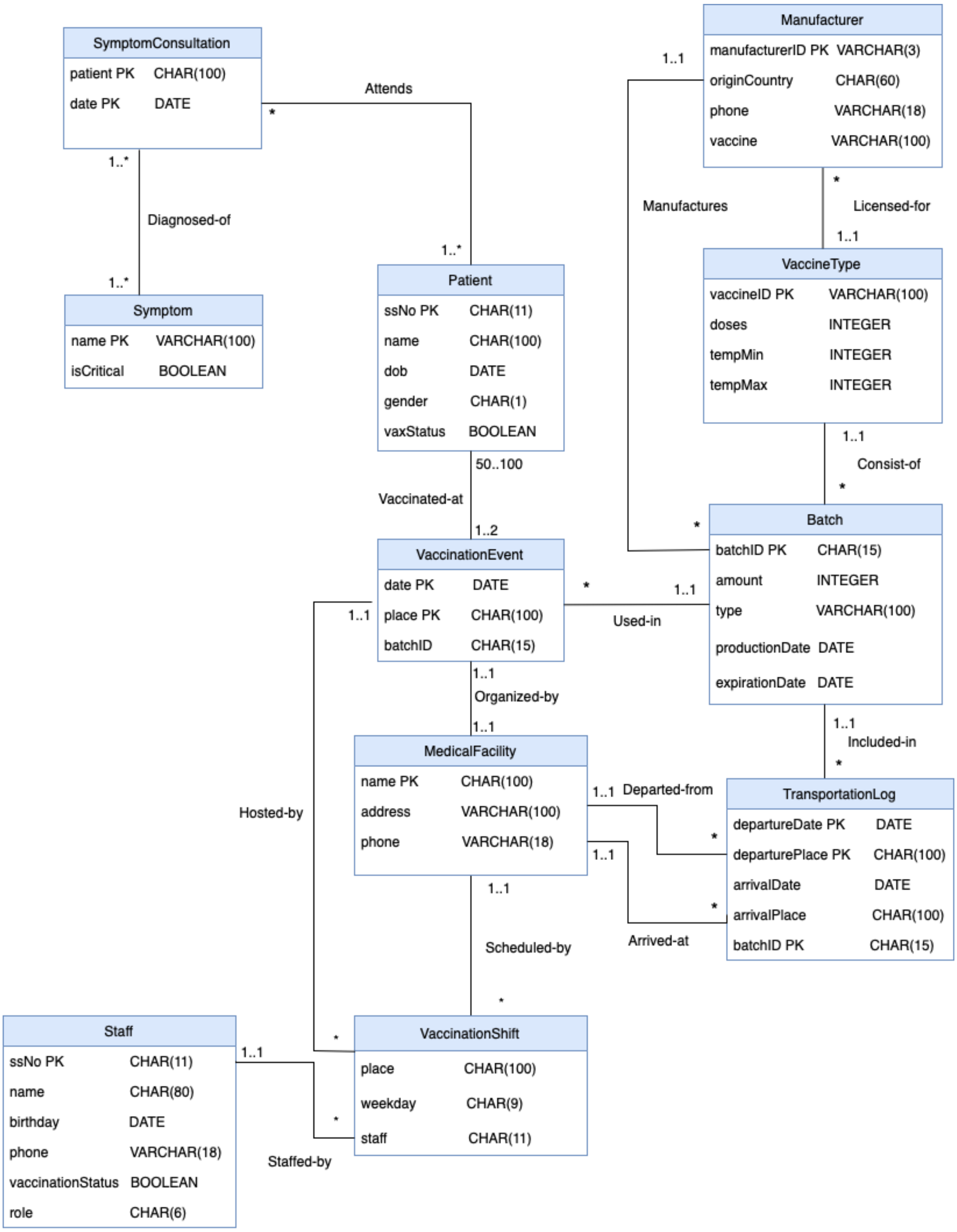
CS-A1153 Databases: Vaccine Distribution Project

Group 9: Bin Choi, Karina Mina, Phuong Hoang, Tommaso Praturlon

Contents

1 Constructing the UML Diagram and Relational Data Model (Part I)

1.1 UML Diagram



1.1.1 Notes about Design Choice

Manufacturer:

- The relation stores information about the manufacturer of vaccines.
- It is assumed that one manufacturer can produce only one vaccine type, and many manufacturers can produce the same vaccine type.

VaccineType:

- *vaccineID* is chosen as PK for `VaccineType` because it can be used to identify the vaccine type in associated tables.

Batch:

- This class stores the data about received vaccine batches: batch ID, number of vaccines, vaccine type, production and expiration date. *BatchID* is a PK because we assume it is the only unique attribute.

VaccinationEvent:

- This class stores the data about vaccination events: date, place and batch ID. According to the descriptions provided and our assumptions, on the same date there can be events in different places so date cannot be PK, at the same time same place can be used for events on different dates so place can't be PK either, same goes for the batchID, which can be used in several places and on several dates. However, on the same date in the same place there can only be one batchID, so *date* and *place* are both PKs.

Symptom:

- We first identify which attribute will be the PK for `Symptom` by asking some relevant questions:
  - Can one patient be diagnosed of the same symptom more than once (e.g. breathlessness initially diagnosed as not critical, next visit diagnosed as critical)?
  - Should we instead define the `symptom` relation as a `symptomConsultation` which is a subclass of a new class `Consultation` (which although is unused now, may increase scalability/expandability of this database in the future)?
  - Can we make the assumption that the symptom name locks in the criticality of the symptom?
- After inquiring with the TA and group mates, we realized that the last point is a relevant and fair assumption to make which can reduce the number of redundancies.
- Therefore, relation `Symptom` represents the symptom description/name and their criticality and is referenced by `SymptomConsultation`.

SymptomConsultation:

- This class will handle the data of patients' visit to hospital as result of experiencing symptoms.

- it stores date of diagnosis, name of symptoms, and the patient's ssNo.

Patient:

- There was not much difficulty with designing this relation as it is quite straightforward.
- This relation stores relevant particulars of patients who are receiving the vaccine.

TransportationLog:

- The table contains records about the movemet of the batches between different medical facilities
- For this relation we assume that one batch does not move more than once a day. However, it may be possible that one batch is moved more than once in a week, therefore the selection of the composite PK.
- Another assumption made is that a tuple records only the initial and final place for the trip, not possible intermediate stops. For example, if the batch is moved from A to C through B, only A and C will appear in the tuple.
- One medical facility can move more than one batch on the same day.

MedicalFacility:

- the relation stores information about hospitals and clinics where the vaccine batches are stored and where vaccinations take place.

VaccinationShift:

- This table should contains information about the shifts that are organized at hospitals. Therefore it must be connected with both tables `MedicalFacility` and `Staff`. Because:
  - none of the fields `MedicalFacility` and `weekday` is unique, thereby leading this table to have anomalies, and
  - one staff, who can be a part-time worker, can work on different shifts at the same hospital,

  it would be logical to include *hospital*, *weekday* and *staff* as fields of `VaccinationShift`. Field *staff* should be a foreign key connecting with table `Staff`'s primary key

Staff:

- As the table staff has contained *staff* that will be linking to this table, now we should decide the PK for `Staff`. The best candidate for PK is *ssNo* because it is unique to each personnel.

1.2 Relational Data Model

Manufacturer(manufacturerID, originCountry, phone, vaccineID)
VaccineType(vaccineID, doses, tempMin, tempMax)
Note: The assumption made for the association between `Manufacturer` and `VaccineType` is that many manufacturers can produce one type of vaccine.
Batch(batchID, amount, type, productionDate, expirationDate)
VaccinationEvent(date, place, batchID)
VaccinatedAt(date, place, ssNo)
Patient(ssNo, name, dob, gender, vaxStatus)
SymptomConsulation(patient, date, symptom)
Symptom(name, isCritical)
Note: The assumption made in the final relation `Symptom` is that the symptom name locks in the criticality of the symptom experienced. In other words, we assume that the symptom name (diagnosed and recorded by the doctor) is descriptive of the criticality.
MedicalFacility(name, address, phone)
VaccinationShift(place, weekday, staff)
Staff(ssNo, name, birthday, phone, vaccinationStatus, role)
TransportationLog(batchID, departureDate, departurePlace, arrivalDate, arrivalPlace)

1.3 Assessing the Model

1.3.1 Functional Dependencies

Note: In addition to the FD's explicitly listed below, the full list of FD's include all of the variations (of different combinations of dependants) that can be created by applying *Splitting & Combining FD rules* to the FD's below as explained during lecture.

Manufacturer
manufacturerID → originCoutry, phone, vaccine
phone → manufacturerID, originCountry, vaccine

VaccineType
vaccineID → doses, tempMin, tempMax

Batch
batchID → amount, type, productionDate, expirationDate

VaccinationEvent
date, place → batchID

VaccinatedAt
No FD as all attributes are PK.

Patient
ssNo → name, dob, gender, vaxStatus

SymptomConsultation
No FD as all attributes are PK.

Symptom
name → isCritical

MedicalFacility
name → phone, address
phone → name, address

TransportationLog
batchID, departureDate, departurePlace → arrivalDate, arrivalPlace
batchID, arrivalPlace, arrivalDate → departureDate, departurePlace
batchID, departureDate → departurePlace, arrivalPlace, arrivalDate
batchID, arrivalDate → departureDate, departurePlace, arrivalPlace

VaccinationShift
place → staff
place, weekday → staff

Staff
ssNo → name, birthday, phone, vaccinationStatus, role
phone → ssNo, name, birthday, vaccinationStatus, role
name, birthday → ssNo, phone, vaccinationStatus, role

1.3.2 Redundancy and Anomaly

Manufacturer
The relation has very little redundancy. The case in which the most tuples would be similar to each other would be when one manufacturer produces many different vaccines. However, given a realistic scenario in which only few vaccines are available, the redundancy would not cause problems.

VaccineType
There is little redundancy in this relation.

Batch
There is redundancy in this relation. Quite often the only variation between tuples would be the *batchID*.

VaccinationEvent
There is little to no redundancy in this relation (only date or place can sometimes appear several times).

VaccinatedAt
There is redundancy in this relation. Most often the only variation between tuples would be the *ssNo* because around 50-100 people are being vaccinated during one vaccination event (meaning same place and date attributes).

Patient
There is little to no redundancy in the relation Patient.

SymptomConsultation
There is some redundancy in this relation. For instance, if one patient is diagnosed of three different symptoms in one consultation, this will result in three different tuples with identical values in all attribute fields except for `symptom` field. This is a bearable amount of redundancy and there is not many ways to better represent the data.

Symptom
There is little to no redundancy in the relation Symptom.

TransportationLog
There is redundancy in the case in which one medical facility moves more than one batch on the same day to the same hospital or clinic. In this case the only variation between tuples would be the *batchID*. However, assuming that very few batches are moved at the same time from the same facility, the relation is acceptable.

MedicalFacility
In this relation there are not apparent anomalies or redundancies.

VaccinationShift
place and *weekday* are repeated because of *staff.* This is a redundancy that can cause deletion and update anomalies. However this is an acceptable redundancy because when we consider the functional dependencies (see **1.3.1**) and BCNF (see **1.3.3**), if we split this relation into smaller relations with available FDs, we will have two relations: one with *place* and *staff* and one with *place* and *weekday.* These relations will be too unnecessarily granular to portray vaccination shifts.

Staff
There is little redundancy in this relation.

1.3.3 Boyce-Codd Normal Form

Manufacturer:

- **R(manufacturerID, originCountry, phone, vaccine)**:
  - Considering the FD phone → manufacturerID, originCountry, vaccine, here *phone* is a key therefore also a superkey
  - Therefore this relation is in BCNF

VaccineType:

- **R(vaccineID, doses, tempMin, tempMax)**:
  - FD *vaccineID → doses, tempMin, tempMax* has *vaccineID* as a superkey
  - Therefore this relation is in BCNF

Batch:

- **R(batchID, amount, type, productionDate, expirationDate)**:
  - Consider FD batchID → amount, type, productionDate, expirationDate. The LHS: *batchID* is a superkey of the relation
  - Therefore this relation is in BCNF

VaccinationEvent:

- **R(date, place, batchID)**:
  - Consider FD date, place → batchID. The LHS: *date, place* is a superkey of the relation
  - Therefore this relation is in BCNF

VaccinatedAt:

- **R(date, place, ssNo)**:
  - There are no FD's for this relation.
  - Therefore this relation is in BCNF

Patient:

- **R(ssNo, name, dob, gender)**:
  - Consider FD ssNo → name, dob, gender, vaxStatus. The LHS: *ssNo* is a superkey of the relation
  - Therefore this relation is in BCNF

Symptom:

- **R(name, isCritical)**:
  - Consider FD name → isCritical. The LHS: *name* is a superkey of the relation
  - Therefore this relation is in BCNF

SymptomConsultation:

- **R(patient, date, symptom)**:
  - There are no FD's for this relation.
  - Therefore this relation is in BCNF

MedicalFacility:

- **R(name, phone, address)**:
  - Considering the FD phone → name, address *phone* is a key therefore also a superkey
  - Therefore this relation is in BCNF

TransportationLog:

- **R(batchID, departureDate, departurePlace, arrivalDate, arrivalPlace)**:
  - Considering the FD batchID, arrivalPlace, arrivalDate, the left-hand side of the dependency is a superkey
  - the same is valid of all the other nontrival FDs
  - Therefore this relation is in BCNF

VaccinationShift:

- **R(place, weekday, staff)**:
  - FD *place* → *staff* holds in this relation but the left side is not a superkey.
  - Therefore this relation is not in BCNF.
- **R1(place, staff)**:
  - FD *place* → *staff* has *place* as a superkey in this relations
  - Therefore this relation is in BCNF.
- **R2(place, weekday)**:
  - There is no FD in this relation
  - Therefore this relation is in BCNF.

Staff:

- **R(ssNo, name, birthday, phone, vaccinationStatus, role)**:
  - FD *ssNo* → *name, birthday, phone, vaccinationStatus, role* has *ssNo* as a superkey
  - FD *phone* → *ssNo, name, birthday, vaccinationStatus, role* has *phone* as a superkey
  - FD *name, birthday* → *ssNo, phone, vaccinationStatus, role* has *name, birthday* as a composite superkey
  - Therefore this relation is in BCNF

2 Modelling the Vaccine Distribution in Finland (Part II)

2.1 Database changes

Overall, we updated many data types in Part II for the purpose of consistency. All attributes with variable length became varchar(100) e.g name, location. All attributes with fixed number of characters became char(*), where * is the number of characters e.g ssNo is always char(11).

Staff
According to the feedback from previous assignment, *Staff* has been added another column of *place* where the staff works. A foreign key was also added to connect relation *Staff* with *MedicalFacility*

Batch
A new one-to-many relation was added **is-stored-in** between *Batch* and *MedicalFacility* with * on *Batch*, so location was added as an attribute and FK to relation *Batch*. We also noticed that relation *Batch* was missing manufacturer from *Manufacturer* among its attributes although there was a one-to-many association between these two relations already in our UML diagram in Part I, so we added it along with location when creating tables in database.

2.2 Data cleaning

The data cleaning process can be described shortly as:

- First, we iterate through the whole Excel and return a dictionary that contains key as sheet name and value as sheet content. Each of these values is a pandas dataframe.
- Second, we iterate through this dictionary's values and clean them using functions that we have made.

The whole process's codes in detail can be found in file process_raw.py
Below are the data errors found while inserting into relations the content provided and what we have done to handle the errors.

2.2.1 Empty columns and rows

There were some empty columns and rows from the Excel files. This was handled by .dropna() function applied to both axes. Just in case we also trim spaces from values.

```
def trim_df(df=None):
    if df is None:
        print("Dataframe must be passed")
    else:
        df.dropna(inplace=True,how='all', axis='columns')
        df.dropna(inplace=True,how='all')
        df.applymap(lambda x: x.strip() if isinstance(x,str) else x)
```

2.2.2 Social security number in wrong format

The social security numbers listed are not in the correct format of how Finnish social security number should be. Strings were simply split at their positions and pasted together to return the correct format.

```
def process_ssNo(df):
    for col in df.columns.values:
        if check_is_ssNo(col) is True:
            df[[’first’,’second’]] = df[col].str.split(’-’, expand=True)

            if len(df[col][0]) == 13:
                    df[’year’] = df[’first’].str[2:4]
                    df[’month’] = df[’first’].str[4:6]
                    df[’day’] = df[’first’].str[6:]
            else:
                df[’year’] = df[’first’].str[:2]
                df[’month’] = df[’first’].str[2:4]
                df[’day’] = df[’first’].str[4:]

            df[col] = df[’day’]+ df[’month’] + df[’year’] + ’-’ + df[’second’]

            df= df.drop([’first’,’second’, ’day’, ’month’,
                ’year’],axis=1,inplace=True)
```

check_is_ssNo() is a helper function to determine whether the column contains social security numbers.

2.2.3 Fix boolean type

The format of boolean in Excel file was 1s and 0s. We check if columns only has 1s and 0s and turn them into type bool

```
for col in df.columns.values:
            if df[col].isin([1,0]).all():
                df[col] = df[col].astype(bool)
```

2.2.4 Wrong dates format and ValueError: day out of month range

We noticed there are some date errors in sheet *Diagnosis*. To handle the error, we wrote function handle_dates and apply to all columns that we check could be dates. The idea of this function is:

- First, we iterate through the rows in the column. If the row is integer then we update row with date. Formula is date 30 December 1899 + value of the row.

- Second, we check if the date is actually date by trying to extract from row value year value. Doing this instead of the normal checking of the full date format is because we simply do not care what exactly is the format of dates. There are two reasons why we do not care:

  - Even if pandas could not pass the data as dtype_datetime64_any due to some errors, it always recognizes date format from Excel and pass in as `YYYY-MM-DD and maybe something else after that` object.
  - Luckily, postgres is also generally easy with date type and will pass in any date format as date, as long as it is not a timestamp, which does not appear in our case.

- If the extraction return error, we catch this error and handle by simply splitting the object at day position (because object is always in format `YYYY-MM-DD ....`) and subtract by 1. Then we update the row with the full string we just fixed.

```
def handle_dates(df, col_name):
    for row in df[col_name]:
        if isinstance(row, numbers.Number):
            df[col_name].mask(df[col_name] == row,datetime.date(1899, 12,30) +
                datetime.timedelta(days=row) , inplace=True)
        else:
            try:
                row.strftime(’%Y’)
            except Exception:
                df[col_name].mask(df[col_name] == row, str(row)[:8] +
                    str(int(str(row)[8:10]) - 1) + str(row)[10:], inplace=True)
```

2.2.5 Extra columns in the data compared to our original UML design

This is not a data error per se but is included because it interferes with inserting the data into sql relation smoothly and we cannot edit the raw data to fit our UML. This is handled by getting relations' number of columns from `information_schema` and bind the same amount of variables and dataframe columns in our insert sql query. This step was included as a class `Postgres` method instead of in process_raw.py because it has to do more on Postgres side.

```
def execute_insert(self, df=None, sql_table=None):
        if df is None or sql_table is None:
            print("All parameters must be passed.")
        else:
            try:
                # get amount of columns that we define in our tables
                sql_table = sql_table.lower()
                self.execute_single_sql(’select count(*) as column_count from
                    information_schema."columns" where table_schema = \’public\’
                    and table_name = %s’, (sql_table,))
                sql_cols = self.cursor.fetchone()[0]

                # insert only the first amount of columns that we define
                sql = "insert into " + sql_table + " values(" +
                    bind_string(int(sql_cols)) + ")"

                # truncate table
                trunc_sql = ’truncate table ’ + sql_table + ’ cascade’
                self.execute_single_sql(trunc_sql)

                df = df[df.columns[:sql_cols]]

                for i in range(len(df.index)):
                    self.execute_single_sql(sql, df.values.tolist()[i],
                        commit=True)

            except (Exception, Error) as e:
                raise
```

2.3 SQL queries

2.3.1 Query #1

This query is to find all staff members who work at vaccination on May 10, 2021. This was achieved by joining relations *Staff*, *VaccinationEvent* and *VaccinationShift*. Because there is only *date* in *VaccinationEvent*, *weekday* and *date* must be turned into numbers to join.

```sql
SELECT Staff.* FROM Staff JOIN
    (SELECT staff, place,
    CASE WHEN weekday = 'Sunday' THEN 0
    WHEN weekday = 'Monday' THEN 1
    WHEN weekday = 'Tuesday' THEN 2
    WHEN weekday = 'Wednesday' THEN 3
    WHEN weekday = 'Thursday' THEN 4
    WHEN weekday='Friday' THEN 5
    WHEN weekday = 'Saturday' THEN 6 END AS dow_no
    FROM VaccinationShift) shift
ON Staff.ssNo = shift.staff
JOIN (SELECT date, extract(dow from date) AS wd, place FROM
   vaccinationevent) event
ON shift.dow_no = event.wd and shift.place = event.place
WHERE event.date = '2021-05-10';
```

The reference results for the latter query are reported in Table 1.

2.3.2 Query #2

The query below is to find the doctors that are available on Wednesdays in Helsinki. The author interpreted the request as finding the doctorS who will be working in vaccination event on Wednesday shifts at a hospitals in Helsinki. To achieve this the relations *Staff*, *VaccinationShift* and *MedicalFacility* are joined together.

```sql
SELECT Staff.name FROM Staff JOIN
VaccinationShift ON Staff.ssNo = VaccinationShift.staff
JOIN MedicalFacility ON VaccinationShift.place = MedicalFacility.name
```

```
WHERE VaccinationShift.weekday = 'Wednesday' AND Staff.role = 'doctor'
    AND MedicalFacility.address LIKE '%HELSINKI%';
```

The reference results for the latter query are reported in Table 2.

2.3.3 Query #3

The intent of this query was to state the current location and the last location of a vaccine batch and compare them for consistency check. The solution applied involved a subquery which returned for each *batchID* in the relation *TransportationLog* the date of the last movement of the batch, along with its ID. This result is later intersected with the *TransportationLog* table for the ID and date and from this query the last location is taken, while from the realtion *Batch* the current location and ID are kept.

```
SELECT T1.batchID,
    location AS currentLocation,
    T1.arrivalPlace AS lastLocation
FROM Batch,
    TransportationLog AS T1
   INNER JOIN (
      SELECT batchID, MAX(arrivalDate) AS lastDate
      FROM TransportationLog
      GROUP BY batchID
   ) AS T2
   ON T1.batchID = T2.batchID
      AND T1.arrivalDate = lastDate
WHERE Batch.batchID = T1.batchID
;
```

The reference results for the latter query are reported in Table 3.
The second part of the query asked to list the *batchID* where the current location was not consistent with the report in the transportation log, and the phone number of the facility where it was supposed to be found. For this reason the following query makes use of the previous one (3a) and compares the *currentLocation* and *lastLocation*. When these are different the query returns the desired output. The results can be referenced in Table 4.

```
SELECT Q1.batchID, phone
FROM MedicalFacility,
   (SELECT T1.batchID,
       location AS currentLocation,
       T1.arrivalPlace AS lastLocation
   FROM Batch,
       TransportationLog AS T1
       INNER JOIN (
       SELECT batchID,
           MAX(arrivalDate) AS lastDate
       FROM TransportationLog
       GROUP BY batchID
   ) AS T2
   ON T1.batchID = T2.batchID
      AND T1.arrivalDate = lastDate
   WHERE Batch.batchID = T1.batchID
) AS Q1
WHERE Q1.currentLocation != Q1.lastLocation
   AND Q1.lastLocation = MedicalFacility.name;
```

2.3.4 Query #4

With this query the intent is to find all patients with critical symptoms diagnosed later than May 10, 2021 and match this data with the data about the vaccines the patient has been given (such as batches of the vaccines, the type of the vaccine, the date the vaccine was given, and the location of the vaccination). Two subqueries were used to retrieve the needed information. From the relation *Symptom* only critical symptoms were selected, then all the information regarding the patient has been selected from relations *Patient* and *SymptomConsultation* where also the filtering for date and symptom was applied. Lastly, the outer query selects patient's name and ssNo from the results generated by inner queries as well as it also selects the vaccination related data such as batchID, vaccine type, vaccination date and location from relations *VaccinationEvent*, *VaccinatedAt* and *Batch*. The query produces 0 results, which are displayed in Table 5. The output should be correct because there were no critical symptoms diagnosed after May 10,2021. If we use the same query and change the date filtering for earlier than May 10, 2021, then it generates results properly, which means that the query itself works as intended and produces desired results.

```
SELECT P.ssNo, P.name, VaccinationEvent.batchID, Batch.type,
   VaccinatedAt.date, VaccinatedAt.place
FROM (SELECT Patient.name, Patient.ssNo, SymptomConsultation.symptom
       FROM Patient, SymptomConsultation
       WHERE Patient.ssNo = SymptomConsultation.patient AND
          SymptomConsultation.date > '2021-05-10'
       GROUP BY name, ssNo, symptom
       HAVING SymptomConsultation.symptom IN(
           SELECT name FROM Symptom WHERE isCritical is True)
) AS P, VaccinationEvent, Batch, VaccinatedAt
WHERE VaccinatedAt.ssNo=P.ssNo AND VaccinationEvent.batchID =
   Batch.batchID AND
VaccinationEvent.place=VaccinatedAt.place AND
   VaccinationEvent.date=VaccinatedAt.date
;
```

2.3.5 Query #5

With this query the intent is to create a view for patients with additional column "vaccinationStatus". This column takes the value 1 if the patient has attended enough vaccinations, and 0 otherwise. Although all of the vaccine types in the data provided require two doses, I do not think it is appropriate to check that each patient received two doses (irregardless of the vaccine type) as new vaccine types could be added to the database in the future. Then, the query will be rendered useless. Therefore, I designed my query such that it considers the vaccine type that the patient has received and checks whether they received the necessary number of dosages. The results are displayed in Table 6.

```sql
CREATE VIEW patientVaxStatus AS (
SELECT
   ssno,
   name,
   dob,
   gender,
   CASE
      WHEN vaxcount = doses THEN 1
      WHEN vaxcount != doses THEN 0
   END vaxStatus
FROM
   patient
NATURAL JOIN (
   SELECT
      *
   FROM
      (
      SELECT
         ssno,
         COUNT(*) AS vaxCount
      FROM
         vaccinatedat v
      GROUP BY
         ssno) AS t1
   NATURAL JOIN (
      SELECT
         DISTINCT ssno,
         doses
      FROM
         (
         SELECT
            ssno,
            type
         FROM
            (vaccinatedat
         NATURAL JOIN vaccinationevent) AS v,
            batch b
         WHERE
            v.batchid = b.batchid) AS t1
      INNER JOIN vaccinetype ON
         vaccineid = type) AS t2)
   AS T2
);
-- to see all tuples of the view...
```

```
SELECT * FROM patientVaxStatus;
```

2.3.6 Query #6

With this query the intent is to find the total number of vaccines stored in each medical facility and the number of distinct vaccine types available. From the relation *Batch* it is possible to retrieve all these information with the following query, grouping by medical facility. The results are displayed in Table 7.

```
SELECT location,
    SUM(amount) AS totVax,
    COUNT(DISTINCT type) AS vaxTypes
FROM Batch
GROUP BY location
;
```

2.3.7 Query #7

With this query the intent is to find the average frequency of different symptoms diagnosed for each vaccine type/name. The symptom should not be considered to be caused by the vaccine, if it has been diagnosed before the patient got the vaccine. If a patient has received two different types of vaccines before the diagnosis of the symptom, the symptom should be counted once for both of the vaccines. The key to coming up for this query was creating the intermediate output `T1` and picking out the relevant statistics from there. Also, we were advised to make the assumption that all occurrences of a symptom report is counted for more accurate result (e.g. same symptom diagnosis of one patient is counted twice, one patient can be diagnosed of multiple symptoms, etc...). The results are displayed in Table 8.

```
WITH
T1 AS (
SELECT
   v.ssno AS pid,
   name AS VaccineName,
   symptom
FROM
   (
   vaccinatedat v
INNER JOIN VaccinationEvent e ON
   e.place = v.place
   and e.date = v.date
NATURAL JOIN Batch
INNER JOIN vaccinetype ON
   vaccinetype.vaccineid = Batch.type
   ),
   symptomconsultation c
WHERE
   c.patient = v.ssno
   AND c.date > v.date
),
T2 AS (
SELECT
   VaccineName,
   COUNT(*) AS count
FROM
   T1
GROUP BY
   VaccineName
),
T3 AS (
SELECT
   VaccineName,
   symptom,
   COUNT(*) AS count
FROM
   T1
GROUP BY
   VaccineName,
   symptom
)
SELECT
   T2.VaccineName,
   T3.symptom,
   ROUND(1.0 * T3.count / T2.count, 4) AS Frequency
FROM
   T3
INNER JOIN T2 ON
   T3.VaccineName = T2.VaccineName;
```

Note: to identify by vaccine type (id) rather than vaccine name, we simply replace `VaccineName` with `VaxType`. The results are displayed in Table 9.

```
with
T1 as (
select
   v.ssno as pid,
   vaccinetype.vaccineid as VaxType,
   symptom
from
   (
   vaccinatedat v
inner join VaccinationEvent e on
   e.place = v.place
   and e.date = v.date
natural join Batch
inner join vaccinetype on
   vaccinetype.vaccineid = Batch.type
   ),
   symptomconsultation c
where
   c.patient = v.ssno
   and c.date > v.date
),
T2 as (
select
   VaxType,
   COUNT(*) as count
from
   T1
group by
   VaxType
),
T3 as (
select
   VaxType,
   symptom,
   COUNT(*) as count
from
   T1
group by
   VaxType,
   symptom
)
select
   T2.VaxType,
   T3.symptom,
   ROUND(1.0 * T3.count / T2.count, 4) as Frequency
from
   T3
inner join T2 on
   T3.VaxType = T2.VaxType;
```

3 Data Analysis (Part III)

3.1 Task 1

In this question we were asked to connect to postgres database using SQLAlchemy and create a new table called `patientsymptom`. We created this table by reading from an sql query to a pandas dataframe. Then, we use pandas `to_sql` function to make this a postgres table.

```
conn_str = ’postgresql+psycopg2://’+ credentials[’user’] +’:’ +
    credentials[’password_parsed’] + ’@’ + credentials[’host’] +’/’ +
    credentials[’database’]

engine = create_engine(conn_str)
conn = engine.connect()

# question 1
conn.execute(’drop table if exists patientsymptom;’)
patient_symptom = pd.read_sql_query(’select ssno, name, dob as
    dateofbirth, gender , symptom, date as diagnosisdate from patient join
    symptomconsultation on patient.ssno = symptomconsultation.patient;’,
    conn)
patient_symptom.to_sql(’patientsymptom’, con=conn, index=True,
    if_exists=’replace’)
```

3.2 Task 2

In this question we were asked to create a new pandas dataframe called `PacientVaccineInfo`. We created this table by reading from an sql query to a pandas dataframe. Then, we use pandas `to_sql` function to make this a postgres table.

```
# same connection method as listed in task 1
# question 2
q2_sql = " ".join(["with t1 as (","select","ssno as
    patientssNo,","v.date,","b.type as vaccinetype",
"from","vaccinatedat v","inner join vaccinationevent v2 on","v.date =
    v2.date","and v.place = v2.place",
"inner join batch b on","v2.batchid = b.batchid),","t2 as
    (","select","patientssNo,","min(date) as date1,",
"case","when count(*) = 1 then null","when count(*) = 2 then
    max(date)","end date2","from","t1","group by",
"patientssNo","order by","patientssNo),","t3 as
    (","select","t1.patientssNo,","date1,","vaccinetype as vaccinetype1,",
"date2","from","t1","inner join t2 on","t1.patientssNo =
    t2.patientssNo","and t1.date = t2.date1),","t4 as (",
"select","t3.patientssNo,","date1,","vaccinetype1,","date2,","t1.vaccinetype
    as vaccinetype2","from","t1",
"inner join t3 on","t1.patientssNo = t3.patientssNo","and t1.date =
    t3.date2)","select","t3.patientssNo,",
"t3.date1,","t3.vaccinetype1,","t4.date2,","t4.vaccinetype2","from","t3","full
    outer join t4 on",
"t3.patientssNo = t4.patientssNo","and t3.date1 = t4.date1;"])

patient_vaccine_info = pd.read_sql_query(q2_sql, conn)
patient_vaccine_info.to_sql('patientvaccineinfo', con=conn, index=True,
    if_exists='replace')
```

3.3 Task 3

In this question we were asked to create two new pandas dataframes from the PatientSymptoms table. We split this table by gender (male or female). Then, we analyzed the individual tables to determine the top three most common symptoms for males and females.

```
# same connection method as listed in task 1
# question 3
patient_symptom_female = pd.read_sql_query('select * from patientsymptom p
    where p.gender = \'F\';', conn)
patient_symptom_male = pd.read_sql_query('select * from patientsymptom p
    where p.gender = \'M\';', conn)

sql_query_to_answer_q3f=" ".join(["with f as (","select * from
    patientsymptom p where p.gender = 'F')","select symptom, count(*) as
    symptomcount","from f","group by symptom","order by symptomcount
    DESC;"])
print("> FEMALE:")
print(pd.read_sql_query(sql_query_to_answer_q3f, conn).head())
# in female patients, the most frequently experienced symptoms are...
# 1. muscle ache  8
# 2. headache     7
# and three-way tie for 3.
#   feelings of illness 4
#   fever              4
#   joint pain         4

sql_query_to_answer_q3m=" ".join(["with m as (","select * from
    patientsymptom p where p.gender = 'M')","select symptom, count(*) as
    symptomcount","from m","group by symptom","order by symptomcount
    DESC;"])
print("> MALE:")
print(pd.read_sql_query(sql_query_to_answer_q3m, conn).head())
# in male patients, the most frequently experienced symptoms are...
# 1. joint pain    10
# 2. muscle ache 7
# and a tie for 3.
#   fever      6
#   headache 6
```

3.4 Task 4

In this question we were asked to create a new pandas dataframe from the table `Patient`. For each tuple, an additional column “ageGroup” was added in the following manner. The use of `case` clause in the query was critical to completing this task.

```
# same connection method as listed in task 1
# question 4
sql_query_to_answer_q4 = " ".join(["select","*,","case","when
    date_part(’year’, AGE(dob))>60 then ’60+’","when date_part(’year’,
    AGE(dob))>40 then ’40-60’","when date_part(’year’, AGE(dob))>20 then
    ’20-40’","when date_part(’year’, AGE(dob))>10 then ’10-20’","when
    date_part(’year’, AGE(dob))>0 then ’0-10’","end ageGroup","from
    patient;"])

patient_age_group_df = pd.read_sql_query(sql_query_to_answer_q4, conn)
print(patient_age_group_df.head)
```

3.5 Task 5

Using the same dataframe that was created in the previous task, a new column describing each patient’s vaccination status was added to it. The statuses are defined as ”0” for not vaccinated, ”1” for vaccinated once, and ”2” for fully-vaccinated. The SQL query below has been used for determining vaccination status of patients. In dataframe NaN values were then replaced by 0, when patients did not receive any doses of vaccine. The results are displayed in Table 10.

```sql
SELECT ssNo, count(*) as VaccinationStatus
FROM Vaccinatedat
GROUP BY ssNo;
```

3.6 Task 6

The solution for this task consists of a for loop which counts the number of people in each age group from the dataframe `vaccination_status_df` containing the information about the age group of each patient vaccinated. Then in `vax_status_df_count` it is counted the number of people for each vaccination status on that age group. Later, for each age group it is computed the total number of vaccinated people and then this number is used to compute the percentage of people for each vaccination status.

Starting from the dataframe of the age group ”0-10”, the other age group dataframes are merged in the for loop to create the resulting Table 11.

```
age_group_str = [’0-10’, ’10-20’, ’20-40’, ’40-60’, ’60+’]
# Age group 0-10
vax_status_df = vaccination_status_df[["agegroup",
    "vaccinationstatus"]].where(vaccination_status_df[’agegroup’] ==
    age_group_str[0]).dropna()
vax_status_df_count = vax_status_df.groupby(’vaccinationstatus’).count()
total = vax_status_df_count.sum()
```

```
old_vax_status_df_perc = vax_status_df_count.div(total)

for age_group in age_group_str[1:]:
   vax_status_df = vaccination_status_df[["agegroup",
      "vaccinationstatus"]].where(vaccination_status_df['agegroup'] ==
      age_group).dropna()
   vax_status_df_count = vax_status_df.groupby('vaccinationstatus').count()
   total = vax_status_df_count.sum()
   new_vax_status_df_perc = vax_status_df_count.div(total)
   # merge old and new dataframes
   old_vax_status_df_perc = pd.merge(old_vax_status_df_perc,
      new_vax_status_df_perc, on="vaccinationstatus", how="outer")

old_vax_status_df_perc.columns = age_group_str
vaxstatus_by_agegroup_df = old_vax_status_df_perc.fillna(0)
print(vaxstatus_by_agegroup_df.head)
```

3.7 Task 7

In this task the reference code is commented below and the reference table is at

```
# Merge PatientSymptoms and PatientVaccineInfo on ssNo
    patient_symptom_freq_df = pd.merge(patient_symptom, patient_vaccine_info,
       how='left', left_on='ssno', right_on='patientssno')
    patient_symptom_freq_df = patient_symptom_freq_df[['ssno', 'symptom',
       'diagnosisdate', 'date1', 'vaccinetype1', 'date2', 'vaccinetype2']]
    # Remove patients that have symptoms and are not vaccinates (e.g no date1,
       vaccinetype1)
    patient_symptom_freq_df =
       patient_symptom_freq_df[patient_symptom_freq_df['date1'].notna()]
    # Remove patients who have been vaccinated but with symptoms prior to a
       vaccine dose
    # The condition > is because the date format is string therefore
       2000-01-10 > 2000-01-20, an improvement could be to use datetime
    patient_symptom_freq_df =
       patient_symptom_freq_df[patient_symptom_freq_df.diagnosisdate >
       patient_symptom_freq_df.date1]
    # Leave only the vaccine causing the symptom
    # Iterate over the rows
    df = patient_symptom_freq_df
    for ind in df.index:
       if not pd.isnull(df.at[ind, 'date2']): #if there is a second dose
           if df.at[ind, 'diagnosisdate'] >= df.at[ind, 'date2']: #if the
              symptom is after the second dose
              # Substitute the second dose in the first dose
              df.loc[ind, 'vaccinetype1'] = str(df.at[ind, 'vaccinetype2'])

    # rename column vaccinetype1 as vaxCause
    df.rename(columns={"vaccinetype1":"vaxCause"}, inplace=True)
    patient_symptom_freq_df = df.drop(['date2', 'vaccinetype2'], axis=1)
    # Count the number of symptoms for each vaccine type
```

```
patient_symptom_freq_df = patient_symptom_freq_df[["symptom", "vaxCause"]]
patient_symptom_freq_df = pd.crosstab(patient_symptom_freq_df.symptom,
    patient_symptom_freq_df.vaxCause)
# Compute the total amount of vaccinations for each vaccine type
patient_vaccine_info_date1_df =
    patient_vaccine_info['vaccinetype1'][patient_vaccine_info['vaccinetype1'].notna()]
patient_vaccine_info_date2_df =
    patient_vaccine_info['vaccinetype2'][patient_vaccine_info['vaccinetype2'].notna()]
# Create a summation for each of the two doses
total_col1_df = patient_vaccine_info_date1_df.value_counts()
total_col2_df = patient_vaccine_info_date2_df.value_counts()
# Compute the sum for each vaccine type
totals_vax_df = total_col2_df.add(total_col1_df, fill_value = 0)
patient_symptom_freq_df['V01'] =
    patient_symptom_freq_df['V01'].div(totals_vax_df.loc['V01'])
patient_symptom_freq_df['V02'] =
    patient_symptom_freq_df['V02'].div(totals_vax_df.loc['V02'])
patient_symptom_freq_df['V03'] =
    patient_symptom_freq_df['V03'].div(totals_vax_df.loc['V03'])
# Replace values with strings
df = patient_symptom_freq_df
for ind in df.index:
    for col in df.columns:
      if df.at[ind, col] >= 0.1:
          df.loc[ind, col] = "very common"
      elif df.at[ind, col] >= 0.05 and df.at[ind, col] < 0.1:
          df.loc[ind, col] = "common"
      elif df.at[ind, col] > 0 and df.at[ind, col] < 0.05:
          df.loc[ind, col] = "rare"

patient_symptom_freq_df = df.replace(0, "-")
print(patient_symptom_freq_df)
```

3.8 Task 8

The percentage for the minimal waste of vaccines is 79% and it has been derived by the code below.

```
# Select all vaccination events
    sql_vax_event = """
    SELECT * FROM vaccinationevent
    ;
    """
    vaccinationevent_df = pd.read_sql_query(sql_vax_event, conn)
    # Find expected number of people by the batch chosen for that day
    sql_batch_amount = """
    SELECT batchid, amount FROM batch
    ;
    """
    batch_amount_df = pd.read_sql_query(sql_batch_amount, conn)
    # Find total vaccines available on a location on that day
```

```
sql_vax_avail = """
SELECT SUM(amount), productiondate, expirationdate, location
FROM batch
GROUP BY productiondate, expirationdate, location
ORDER BY location, productiondate, expirationdate
;
"""
vax_avail_df = pd.read_sql_query(sql_vax_avail, conn)
# Find the number of attending patients on that location and day
sql_patients_attending = """
SELECT date, place, count(ssno)
FROM vaccinatedat
GROUP BY date, place
;
"""
patients_per_event_df = pd.read_sql_query(sql_patients_attending, conn)
# Merge VaccinationEvent and Batch on batchID and select amount and find
    number of attending patients
vaccination_event_batch_df = pd.merge(vaccinationevent_df,
    batch_amount_df, on="batchid", how="inner")
batch_amount_patients_df = pd.merge(vaccination_event_batch_df,
    patients_per_event_df, on=[’date’, ’place’], how="outer")
batch_amount_patients_df = batch_amount_patients_df[[’date’, ’place’,
    ’amount’, ’count’]]
batch_amount_patients_df =
    batch_amount_patients_df.rename(columns={’place’:’location’,
    ’amount’:’expectedpeople’, ’count’:’actualpeople’}, inplace=False)
# Merge with the total number of vaccines available on that date
df1 = batch_amount_patients_df
df2 = vax_avail_df
std_df = pd.merge(df1, df2, on="location", how="right")
std_df = std_df[(std_df.date >= std_df.productiondate) & (std_df.date <=
    std_df.expirationdate)]
# Sum all vaccines available on that date
std_df = std_df.groupby([’date’,’location’]).sum()
# Compute the percentage of expected people as the expected people / total
    available vaccines
# Compute the percentage of actual people as the actual people / total
    available vaccines
std_df[’percexp’] = std_df[’expectedpeople’]/std_df[’sum’]
std_df[’percact’] = std_df[’actualpeople’]/std_df[’sum’]
# Mean and standard deviations
mean = std_df[’percexp’].mean().round(3)
std = std_df[’percact’].std().round(3)
perc_for_minimal_waste = mean + std
print(f"Mean: {mean}, Standard Deviation: {std}, Percentage for Minimal
    Waste: {perc_for_minimal_waste}")
```

3.9 Task 9

The total number of vaccinated patients with respect to date has been plotted in this task. The results are displayed in Figure 1.

3.10 Task 10

In this task the nurse with ssNo "19740919-7140" has been tested positive for corona on 15.5.2021 and it was required to find the social security numbers and names of the patients and staff members that the nurse may have met in vaccination events in the past 10 days. The SQL query below has been used for that. Due to the fact that nurses' shifts in the database are set on days of the week and not on dates, and the period that we are looking for is 10 days during which nurse was in contact with other nurses, it is concluded that since it is longer than 7 days the nurse has definitely met everyone working the same shifts as her during the whole week and therefore we do not care about specific weekdays as we would if the period was shorter, e.g 3 days. Therefore, the query is a union of two select statements: one returns the nurses that have been in contact with a sick nurse (worked the same shifts in the same facility), the other returns patients who attended vaccinations in the medical facility that nurse works in and during this 10-day period of time before 15.05. Please note that in the previous part of this project we changed format of nurses' ssNo, so in the query below our format is used: '190974-7140'. The results are displayed in Table 13.

```
SELECT t2.ssNo, Patient.name
From (SELECT distinct ssNo
       From Vaccinatedat
       Where place IN(SELECT place From Vaccinationshift
                     Where staff='190974-7140')
                     AND date <= '2021-05-15'
                     AND date >= (date('2021-05-15')-interval '10' day))
                        as t2, Patient
WHERE t2.ssNo= Patient.ssNo
UNION
SELECT t1.staff, Staff.name
From (SELECT distinct staff
       From Vaccinationshift
       Where place IN(SELECT place from Vaccinationshift
                     Where staff='190974-7140')
                     AND weekday IN (SELECT weekday From
                        Vaccinationshift Where staff=190974-7140') AND
                        Staff !='190974-7140') as t1, Staff
WHERE t1.staff=Staff.ssNo;
```

4 Appendix A

ssno	*name*	*birthday*	*phone*	*role*	vaccination status	*place*
802092-4854	Kaden Tromp	1992-08-02	044-624-1591	nurse	t	Tapiola Health Center
914074-7140	Deon Hoppe	1974-09-19	040-399-1121	nurse	f	Tapiola Health Center
614094-4448	Jordy Hilpert	1994-06-15	044-506-1982	doctor	t	Tapiola Health Center
813063-6581	Jazlyn Schneider	1963-08-12	040-868-2528	nurse	t	Sanomala Vaccination Point
007177-5988	Samir Hills	1977-10-03	040-093-0059	nurse	t	Sanomala Vaccination Point
818088-8027	Haylie Wintheiser	1988-08-17	050-448-8894	nurse	t	Myyrmki Energia Areena
212082-5928	Elena Bartell	1982-02-18	041-938-9451	nurse	t	Myyrmki Energia Areena
222072-1761	Alfreda Champlin	1972-02-23	041-631-1851	nurse	t	Myyrmki Energia Areena

Table 1: Query 1

name
Rosalia Simonis
Shaylee Kris
Hilbert Purdy
Elnora Greenholt

Table 2: Query 2

batchID	*currentLocation*	*lastLocation*
B01	Sanomala Vaccination Point	Sanomala Vaccination Point
B02	Messukeskus	Sanomala Vaccination Point
B03	Myyrmäki Energia Areena	Myyrmäki Energia Areena
B04	Malmi	Malmi
B06	Iso Omena Vaccination Point	Myyrmäki Energia Areena
B07	Myyrmäki Energia Areena	Myyrmäki Energia Areena
B08	Tapiola Health Center	Tapiola Health Center
B12	Sanomala Vaccination Point	Sanomala Vaccination Point
B13	Iso Omena Vaccination Point	Iso Omena Vaccination Point
B15	Malmi	Malmi
B16	Tapiola Health Center	Tapiola Health Center
B17	Myyrmäki Energia Areena	Myyrmäki Energia Areena
B18	Tapiola Health Center	Tapiola Health Center
B21	Iso Omena Vaccination Point	Iso Omena Vaccination Point
B22	Myyrmäki Energia Areena	Myyrmäki Energia Areena
B23	Sanomala Vaccination Point	Sanomala Vaccination Point
B24	Malmi	Malmi
B25	Malmi	Malmi
B27	Myyrmäki Energia Areena	Myyrmäki Energia Areena
B28	Iso Omena Vaccination Point	Iso Omena Vaccination Point
B29	Myyrmäki Energia Areena	Sanomala Vaccination Point
B30	Iso Omena Vaccination Point	Iso Omena Vaccination Point

Table 3: Query 3a

batchID	*phone*
B02	093-105-3153
B06	093-104-5930
B29	093-105-3153

Table 4: Query 3b

ssno	*name*	*batchid*	*type*	*date*	*place*

(0 rows)

Table 5: Query 4

ssno	*name*	*dob*	*gender*	*vaxstatus*
291284-112N	Rodolfo O'Reilly	1984-12-29	M	1
140278-1893	Prof. Erling Morar MD	1978-02-14	F	0
030395-191X	Dr. Simeon Keeling II	1995-03-03	M	0
180273-253D	Dereck Beer	1973-02-18	M	0
141297-2818	Prof. Brice Metz PhD	1997-12-14	M	0
250306-323X	Darlene Brakus	2006-03-25	F	0
140699-395X	Josefa Greenfelder DVM	1999-06-14	M	0
180205-4796	Ms. Hassie Runolfsson PhD	2005-02-18	F	0
270100-4899	Ms. Opal Lang	2000-01-27	F	0
301072-5216	Noah Leuschke	1972-10-30	M	0

040189-753F	Lukas Runolfsdottir V	1989-01-04	M	1
010872-8748	Braeden Hackett	1972-08-01	M	0
011002-957O	John Larkin	2002-10-01	M	0
080679-9686	Gisselle Hilpert	1979-06-08	F	0
050884-1135	Lonzo Collier	1984-08-05	M	1
041174-114G	Marvin Fahey	1974-11-04	M	0
221173-126T	Johanna McClure	1973-11-22	F	0
270906-1438	Zetta Runolfsson	2006-09-27	F	0
150385-155F	Andreanne Jakubowski	1985-03-15	M	0
010615-1657	Mrs. Ophelia Corwin Sr.	2015-06-01	F	0
020209-1778	Alvera Medhurst	2009-02-02	F	0
060193-189U	Julius Marks	1993-01-06	M	0
171170-199K	Leland Moen	1970-11-17	M	0
070314-203V	Rhea Hettinger	2014-03-07	M	0
271103-2165	Mathew Buckridge	2003-11-27	M	0
010897-218B	Jonathan Wyman	1997-08-01	M	0
060788-240U	Prof. Willard Marquardt II	1988-07-06	M	0
181187-242U	Dr. Victor Armstrong	1987-11-18	M	0
290911-252V	Lilly Farrell V	2011-09-29	F	0
301004-267L	Isabell Nader	2004-10-30	F	0
140272-2797	Abbey Schuppe	1972-02-14	F	0
170107-2839	Estrella Johns	2007-01-17	F	0
111275-287B	Taylor Krajcik	1975-12-11	F	1
290706-292C	Marilou Ryan	2006-07-29	F	0
070709-295R	Alysson Jakubowski	2009-07-07	F	0
101090-3010	Emerald Johnson	1990-10-10	F	0
210406-302M	Faustino Barton	2006-04-21	M	1
081180-303B	Reva Waelchi	1980-11-08	F	0
010107-326Q	Orpha Bogisich	2007-01-01	F	0
140508-3385	Dakota Greenfelder	2008-05-14	F	0
100888-358W	Braxton Hane	1988-08-10	M	1
210672-378H	Ms. Alisha Ortiz	1972-06-21	F	0
030579-394M	Kathlyn Moore	1979-05-03	F	0
080593-413K	Mr. Reid Little II	1993-05-08	M	0
290696-4156	Rossie Spinka	1996-06-29	F	0
040619-440B	Prof. Kevon Cummings	2019-06-04	M	0
160409-443L	Aliyah Harber	2009-04-16	M	1
050213-474D	Elenora Sawayn	2013-02-05	F	0
040893-509I	Fay Ryan	1993-08-04	F	0
281187-519R	Flossie Torp	1987-11-28	F	1
220699-5231	Sid Hahn	1999-06-22	M	0
270301-525G	Dr. Mireille Hansen	2001-03-27	M	1
271170-5340	Mrs. Lorena Kreiger	1970-11-27	F	0
121018-5367	Bulah Heidenreich	2018-10-12	M	0
240771-5480	Dashawn Schamberger	1971-07-24	M	0
051081-5518	Miss Charity Powlowski	1981-10-05	F	0
260209-5673	Dr. Lamont Ferry	2009-02-26	M	1
200883-576C	Loyal Hoeger	1983-08-20	M	0

180312-5791	Mathilde Smith	2012-03-18	F	0
010201-5814	Harrison Heaney	2001-02-01	M	0
081204-5838	Dr. Margarette Mertz IV	2004-12-08	F	0
300916-586P	Aiden Volkman	2016-09-30	F	1
020916-592P	Britney Gutmann	2016-09-02	F	0
031101-6045	Mrs. Kailyn Collier DVM	2001-11-03	F	0
191188-6103	Destiny Konopelski PhD	1988-11-19	F	0
070689-6113	Prof. Raphael Prosacco DVM	1989-06-07	M	0
070896-613R	Omer Denesik	1996-08-07	M	0
131282-6162	Tabitha Howe	1982-12-13	M	0
250300-6271	Mariam Ritchie	2000-03-25	F	0
221104-6308	Prof. Demarco Hahn	2004-11-22	M	0
150619-6325	Ms. Nelda Brekke PhD	2019-06-15	F	0
040618-6419	Dr. Freddie Cartwright	2018-06-04	M	0
060500-642P	Cassandra Mayert	2000-05-06	F	0
301102-649D	Dr. Jayson Glover DVM	2002-11-30	M	0
030999-6514	Eldred Blanda	1999-09-03	F	0
061088-6543	Trycia Jaskolski	1988-10-06	F	0
060406-686D	Dovie West	2006-04-06	F	0
200411-6983	Nellie Nitzsche	2011-04-20	F	0
040893-7021	Prof. Harrison Toy	1993-08-04	M	0
080776-708H	Maia Towne II	1976-07-08	M	0
281285-732X	Shanna Osinski	1985-12-28	F	0
180321-737O	Corine Hane	2021-03-18	F	1
100385-787I	Cristal Borer	1985-03-10	M	0
020609-7898	Hillard Boehm V	2009-06-02	M	0
131104-8113	Prof. Yessenia Dooley Jr.	2004-11-13	F	0
180509-869W	Spencer Kunde	2009-05-18	M	0
221274-8947	Devon Nicolas	1974-12-22	M	0

Table 6: Query 5: created view is shown using `select` query (after view creation)

location	*totVax*	*vaxTypes*
Iso Omena Vaccination Point	65	3
Malmi	65	3
Messukeskus	120	3
Myyrmäki Energia Areena	85	3
Sanomala Vaccination Point	40	2
Tapiola Health Center	55	2

Table 7: Query 6

vaccinename	*symptom*	*frequency*
Comirnaty	high fever	0.0500
AstraZeneca	itchiness near injection	0.0750
AstraZeneca	muscle ache	0.2000
AstraZeneca	joint pain	0.1500
AstraZeneca	warmth near injection	0.0750
Moderna	feelings of illness	0.1667
Comirnaty	fever	0.1500
Comirnaty	diarrhea	0.1500
Comirnaty	pain near injection	0.0500
Moderna	chills	0.0417
AstraZeneca	inflammation near injection	0.0250
AstraZeneca	diarrhea	0.0250
Moderna	lymfadenopathy	0.0833
Moderna	headache	0.0417
Moderna	nausea	0.0833
Comirnaty	muscle ache	0.1500
AstraZeneca	fatigue	0.0250
Moderna	fever	0.0833
Moderna	high fever	0.0417
Comirnaty	joint pain	0.1000
AstraZeneca	feelings of illness	0.0250
AstraZeneca	headache	0.1500
Moderna	fatigue	0.0417
AstraZeneca	blurring of vision	0.0250
Moderna	joint pain	0.1667
Comirnaty	inflammation near injection	0.0500
AstraZeneca	nausea	0.0750
AstraZeneca	vomiting	0.0250
AstraZeneca	high fever	0.0500
AstraZeneca	fever	0.0750
Comirnaty	chest pain	0.0500
Comirnaty	fatigue	0.0500
Moderna	muscle ache	0.2083
Moderna	vomiting	0.0417
Comirnaty	headache	0.2000

Table 8: Query 7

5 Appendix B

vaxtype	*symptom*	*frequency*
V03	joint pain	0.1000
V03	chest pain	0.0500
V01	vomiting	0.0250
V01	warmth near injection	0.0750
V02	chills	0.0417
V01	headache	0.1500
V03	high fever	0.0500
V01	diarrhea	0.0250
V02	fever	0.0833
V02	high fever	0.0417
V02	nausea	0.0833
V02	muscle ache	0.2083
V01	high fever	0.0500
V03	fever	0.1500
V02	headache	0.0417
V02	joint pain	0.1667
V01	blurring of vision	0.0250
V01	muscle ache	0.2000
V03	headache	0.2000
V01	itchiness near injection	0.0750
V02	vomiting	0.0417
V01	feelings of illness	0.0250
V01	joint pain	0.1500
V03	muscle ache	0.1500
V03	inflammation near injection	0.0500
V01	fever	0.0750
V03	pain near injection	0.0500
V03	diarrhea	0.1500
V02	fatigue	0.0417
V01	inflammation near injection	0.0250
V02	feelings of illness	0.1667
V01	fatigue	0.0250
V03	fatigue	0.0500
V02	lymfadenopathy	0.0833
V01	nausea	0.0750

Table 9: Query 7b: Using a slightly different SQL query, we can compute query 7 to identify by vaccine type

ssno	name	dob	gender	agegroup	vaccinationstatus
291284-112N	Rodolfo O'Reilly	1984-12-29	M	20-40	2
140278-1893	Prof. Erling Morar MD	1978-02-14	F	40-60	1
030395-191X	Dr. Simeon Keeling II	1995-03-03	M	20-40	1
180273-253D	Dereck Beer	1973-02-18	M	40-60	1
141297-2818	Prof. Brice Metz PhD	1997-12-14	M	20-40	1
250306-323X	Darlene Brakus	2006-03-25	F	10-20	1
011202-3734	Prof. Raymond Beahan	2002-12-01	M	10-20	0
140699-395X	Josefa Greenfelder DVM	1999-06-14	M	20-40	1
180205-4796	Ms. Hassie Runolfsson PhD	2005-02-18	F	10-20	1
270100-4899	Ms. Opal Lang	2000-01-27	F	20-40	1
301072-5216	Noah Leuschke	1972-10-30	M	40-60	1
040189-753F	Lukas Runolfsdottir V	1989-01-04	M	20-40	2
010872-8748	Braeden Hackett	1972-08-01	M	40-60	1
011002-957O	John Larkin	2002-10-01	M	10-20	1
080679-9686	Gisselle Hilpert	1979-06-08	F	40-60	1
050884-1135	Lonzo Collier	1984-08-05	M	20-40	2
041174-114G	Marvin Fahey	1974-11-04	M	40-60	1
221173-126T	Johanna McClure	1973-11-22	F	40-60	1
270906-1438	Zetta Runolfsson	2006-09-27	F	10-20	1
221080-152O	Dane Barrows I	1980-10-22	F	40-60	0
150385-155F	Andreanne Jakubowski	1985-03-15	M	20-40	1
010615-1657	Mrs. Ophelia Corwin Sr.	2015-06-01	F	0-10	1
020209-1778	Alvera Medhurst	2009-02-02	F	10-20	1
271180-1830	Ms. Ivy Nolan	1980-11-27	F	40-60	0
060193-189U	Julius Marks	1993-01-06	M	20-40	1
030895-1903	Lyric Funk	1995-08-03	M	20-40	0
171170-199K	Leland Moen	1970-11-17	M	40-60	1
070314-203V	Rhea Hettinger	2014-03-07	M	0-10	1
271103-2165	Mathew Buckridge	2003-11-27	M	10-20	1
010897-218B	Jonathan Wyman	1997-08-01	M	20-40	1
250400-224A	Christian McGlynn	2000-04-25	M	20-40	0
010713-235N	Arnold Medhurst	2013-07-01	M	0-10	0
060788-240U	Prof. Willard Marquardt II	1988-07-06	M	20-40	1
181187-242U	Dr. Victor Armstrong	1987-11-18	M	20-40	1
021270-2506	Godfrey Haley Sr.	1970-12-02	M	40-60	0
290911-252V	Lilly Farrell V	2011-09-29	F	0-10	1
160415-255C	Jacklyn Padberg	2015-04-16	F	0-10	0
301004-267L	Isabell Nader	2004-10-30	F	10-20	1
140272-2797	Abbey Schuppe	1972-02-14	F	40-60	1
170107-2839	Estrella Johns	2007-01-17	F	10-20	1
111275-287B	Taylor Krajcik	1975-12-11	F	40-60	2
290706-292C	Marilou Ryan	2006-07-29	F	10-20	1
070709-295R	Alysson Jakubowski	2009-07-07	F	10-20	1
101090-3010	Emerald Johnson	1990-10-10	F	20-40	1
210406-302M	Faustino Barton	2006-04-21	M	10-20	2
081180-303B	Reva Waelchi	1980-11-08	F	40-60	1
270371-313B	Prof. Ewell Conn	1971-03-27	M	40-60	0
010107-326Q	Orpha Bogisich	2007-01-01	F	10-20	1
220905-330C	Mrs. Bulah Grant	2005-09-22	F	10-20	0
140508-3385	Dakota Greenfelder	2008-05-14	F	10-20	1
110886-3456	Syble Howe	1986-08-11	F	20-40	0
020509-3556	Mr. Dereck Brekke IV	2009-05-02	M	10-20	0
100888-358W	Braxton Hane	1988-08-10	M	20-40	2

VaccinationStatus	*0-10*	*10-20*	*20-40*	*40-60*	*60+*
0	0.555556	0.351351	0.346154	0.500000	0.0
1	0.370370	0.567568	0.538462	0.470588	0.0
2	0.074074	0.081081	0.115385	0.029412	0.0

Table 11: Task 6

vaxCause	*V01*	*V02*	*V03*
symptom			
blurring of vision	rare	-	-
chest pain	-	-	rare
chills	-	rare	-
diarrhea	rare	-	common
fatigue	-	rare	rare
feelings of illness	-	very common	-
fever	common	common	common
headache	very common	rare	very common
high fever	-	rare	rare
inflammation near injection	rare	-	rare
itchiness near injection	common	-	-
joint pain	very common	very common	common
lymfadenopathy	-	common	-
muscle ache	very common	very common	common
nausea	common	common	-
pain near injection	-	-	rare
vomiting	rare	rare	-
warmth near injection	common	-	-

Table 12: Task 7

ssno	*name*
220191-1693	Ashley Konopelski
180321-737O	Corine Hane
020892-4854	Kaden Tromp
140508-3385	Dakota Greenfelder
200883-576C	Loyal Hoeger
160409-443L	Aliyah Harber
290696-4156	Rossie Spinka
050213-474D	Elenora Sawayn
150393-7195	Jeromy McKenzie
210406-302M	Faustino Barton
201083-4745	Madisyn Shanahan
150694-4448	Jordy Hilpert
220699-5231	Sid Hahn
121177-4048	Greg Schuppe
040893-509I	Fay Ryan
281187-519R	Flossie Torp
260209-5673	Dr. Lamont Ferry
080593-413K	Mr. Reid Little II
270301-525G	Dr. Mireille Hansen

Table 13: Task 10

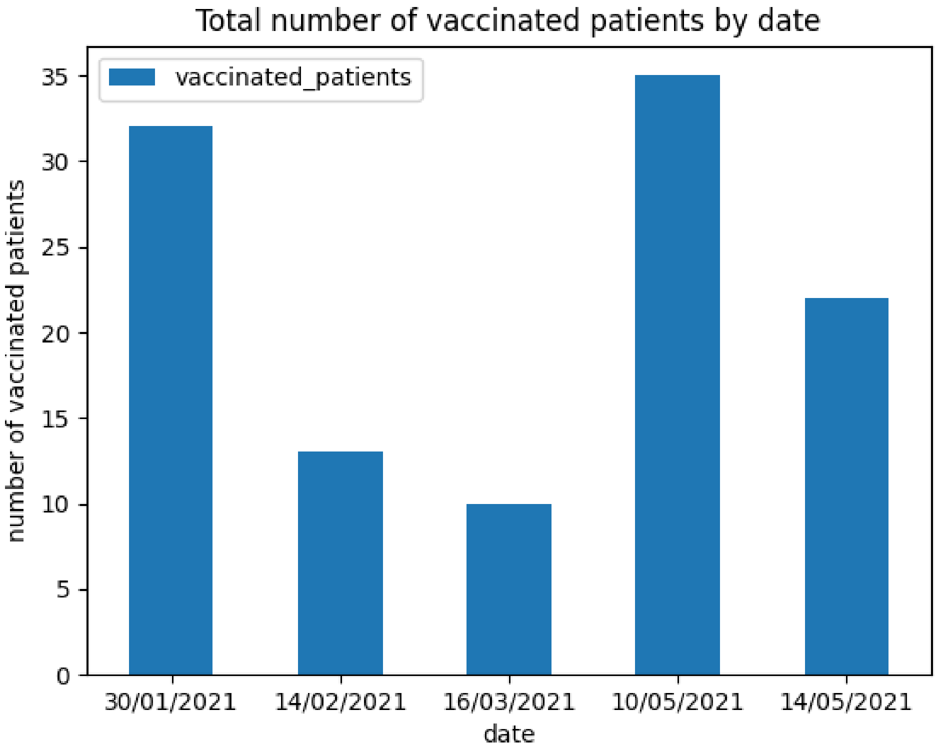


Figure 1: This graph demonstrates total number of vaccinated patients by date.