CS-A1153 - Databases group project
Vaccine distribution

Group 15
Linh Ngo 906502
Rajat Kaul 890906
GuTing Huang 885791
Pouya Adibnezhad 818409
Anna Valldeoriola Cardó 885869

SQL database

The SQL database is created according to the UML diagram created in the previous part, however a few changes were made before populating the tables with data.

- The criticalTemperature column in the Vaccine table is split into two columns tempMin and tempMax to accommodate the date.
- In the VaccineBatch table, the id of the manufacturer is saved as a foreign key rather than the id of the vaccine. Since both of these values were included in the data, this was the only way to not violate the BCNF and still be able to reconstruct the data using table joins.
- As our assumptions regarding the way shifts are assigned to workers and the relationship of the shifts and hospitals were not compatible with the data, the HospitalShift is removed as it was redundant. Since the relationship of the Staff and Shift is many-many, the shift attribute is removed from the Staff table. Instead, the Shift table now contains the social security number of the worker and the weekday. Combinations of these two attributes creates the primary key of the Shift table.
- The location column in the VaccineBatch table has to be added since the initial assumption that the TransportationLog always contains the correct data does not hold on the provided data.

Final version of the schema:

Manufacturer (ID, country, phone, vaccineID)
Vaccine (ID, name, doses, tempMin, tempMax)
VaccineBatch (batchID, amount, manufacturerID, manufDate, location, expiration)
TransportationLog (batchID, arrival, departure, dateArr, dateDep)
Hospital (name, address, phone)

Staff (ssNo, name, dateOfBirth, phone, role, vaccinationStatus, hospital)
Shift (weekday, worker)
VaccinationEvent (date, location, batchID)
Patient (ssNo, name, dateOfBirth, gender)
PatientVaccinationEvent (date, location, patientSsNo)
Symptom (name, criticality)
Diagnosis (patient, symptom, date)

While creating the tables, specific data types were used as much as possible. In some cases, such as the manufacturer ID and vaccine ID, a more general data type (TEXT) is used so that it is possible to add other entries to the table if their id values happen to be longer, e.g. when many more vaccine types are added to the database.

The NOT NULL constraint is used in moderation, and it is only added to columns that are necessarily required for the integrity of the data. For example, the amount of required doses of a vaccine must always be known while it is possible that a vaccine does not have a critical min or max temperature, therefore attribute “doses” in the Vaccine table is not allowed to be null while tempMin and tempMax can.

ON UPDATE CASCADE is explicitly set for almost all instances of foreign keys to make sure the changes to IDs are propagated through the database so that the data can always be queried from the database using joins. Deleting rows that are foreign keys in other tables is not allowed in the table, since the data is highly interconnected. Setting the foreign keys to null can cause obsolete data in the tables. The only exception is the hospital attribute in the Staff table, which is set to null in case a hospital is removed from the database when it is no longer under operation but information of the staff would like to be kept in the database and possibly assigned to other hospitals.

We have also adjusted the name of some attributes for better readability or clarity and referenced foreign keys for all the attributes that relate a table with another, including vaccineID in Manufacturer (with Vaccine table), manufacturerID in VaccineBatch (with Manufacturer), arrival and departure in TransportationLog (old Transportation, with Hospital), hospital in Staff (with Hospital), worker in Shift (with Staff), batchID in VaccinationEvent (with VaccineBatch), patientSsNo (with Patient), location (with Hospital), and date (with VaccinationEvent) in PatientVaccinationEvent, and patient (with Patient) and symptom (with Symptom) in Diagnosis. In addition, we added checks to ensure that the vaccinationStatus, the criticality of a symptom, the role of the staff, and the possible weekdays for Shift have one of the allowed values.

A few of the column names read from the file, such as the location column in Vaccinations sheet, included whitespace which are removed in the Python script. Custom logic is added in the Python code to handle values of the columns containing dates to make sure all the dates added in the database follow the correct format. This was specifically needed for the Diagnosis

table since in the data provided 2 of the rows had dates in incompatible format with the rest of the data.

The file “create_tables.sql” contains the SQL code used to create tables with the corresponding foreign keys and constraints. The file “populate_data.py” contains the Python code that reads the CSV files, cleans them up and adds them to the corresponding tables. Running the “run.py” script runs all the required steps in the correct order and should generate the database file called “vaccination_db.db3”. The queries listed in the next section are included in the “queries.sql” file as well.

Query results

Query 1

```
SELECT DISTINCT Staff.ssNo, Staff.name, Staff.phone, Staff.role,
Staff.vaccinationStatus, VaccinationEvent.location
FROM Staff JOIN VaccinationEvent JOIN Shift
ON Shift.worker = Staff.ssNo AND VaccinationEvent.location = Staff.hospital
WHERE Shift.weekday = "Monday" AND VaccinationEvent.date = "2021-05-10"
```

Description

This query asks for information about staff that work on a specific shift on the date 10/05/2021. Therefore, the relation Staff joins the relation Shift and the relation VaccinationEvent on the attribute social security number of the staffs Staff.ssNo = Shift.worker and VaccinationEvent.location = Staff.hospital. May 10, 2021, is on Monday so the condition for the query is Shift.weekday = “Monday” and VaccinationEvent.date = "2021-05-10". Since each staff member can only work for only one hospital, the location of the vaccination event can be selected from the relation Staff, along with the social security number, phone, role, and vaccination status of the staff.

Result

	ssNo	name	phone	role	vaccinationStatus	location
0	19630812-6581	Jazlyn Schneider	040-868-2528	nurse	1	Sanomala Vaccination Point
1	19720223-1761	Alfreda Champlin	041-631-1851	nurse	1	Myyrmäki Energia Areena
2	19740919-7140	Deon Hoppe	040-399-1121	nurse	0	Tapiola Health Center
3	19771003-5988	Samir Hills	040-093-0059	nurse	1	Sanomala Vaccination Point
4	19820218-5928	Elena Bartell	041-938-9451	nurse	1	Myyrmäki Energia Areena
5	19880817-8027	Haylie Wintheiser	050-448-8894	nurse	1	Myyrmäki Energia Areena
6	19920802-4854	Kaden Tromp	044-624-1591	nurse	1	Tapiola Health Center
7	19940615-4448	Jordy Hilpert	044-506-1982	doctor	1	Tapiola Health Center

Query 2

```
SELECT DISTINCT Staff.name
FROM Staff
JOIN Hospital ON hospital = Hospital.name
JOIN Shift ON worker = ssNo
WHERE address LIKE "% HELSINKI" AND weekday = "Wednesday" AND role =
"doctor"
```

Description

For this query, we have joined the Staff and Hospital based on the hospital name (which is found in the attribute hospital of Staff and the attribute name in Hospital) and these with Shift via the social security number of the employee, found in the attributes worker from Shift and ssNo from Staff. This allowed us to find the doctors (staff with role “doctor”) with the “Wednesday” shift (in the weekday attribute of the Shift table). Also, we needed to join it with the Hospital to be able to extract which hospitals are in Helsinki from the respective addresses (found under the attribute address in Hospital table).

Result

	name
0	Rosalia Simonis
1	Shaylee Kris
2	Hilbert Purdy
3	Elnora Greenholt

Query 3

```
SELECT v.batchID, v.location current_location, t.arrival
last_logged_location
FROM VaccineBatch v
INNER JOIN
(SELECT arrival, MAX(dateArr), batchID
FROM TransportationLog
GROUP BY batchID
) t ON v.batchID = t.batchID;
```

```
SELECT t.batchID, h.phone
FROM VaccineBatch v
INNER JOIN
(SELECT arrival, MAX(dateArr), batchID
FROM TransportationLog
GROUP BY batchID
) t ON v.batchID = t.batchID
INNER JOIN Hospital h ON v.location = h.name
WHERE v.location <> t.arrival;
```

Description

In the first query, the tables involved are VaccineBatch and TransportationLog. We first write a subquery with the TransportationLog table, to create a modified table with information that is more relevant to the problem. Finally, we use an Inner Join with the batchID of the respective tables being used as the common attribute. Now we can select the relevant columns asked for in the problem statement, which are the batch ID, the current location of the batch of vaccines, and their final destination.

The second query uses the information that we generate with the first, involving another Inner Join, this time with the Hospital table, with the name of the hospital being the common attribute. Then, in the Where clause we stipulate the condition of mismatched locations, and finally we can select the required columns, which are the batch ID of the misplaced vaccines and the phone numbers of the hospital/clinic that they were supposed to be at.

Result

	batchID	current_location	last_logged_location
0	B01	Sanomala Vaccination Point	Sanomala Vaccination Point
1	B02	Messukeskus	Sanomala Vaccination Point
2	B03	Myyrmäki Energia Areena	Myyrmäki Energia Areena
3	B04	Malmi	Malmi
4	B06	Iso Omena Vaccination Point	Myyrmäki Energia Areena
5	B07	Myyrmäki Energia Areena	Myyrmäki Energia Areena
6	B08	Tapiola Health Center	Tapiola Health Center
7	B12	Sanomala Vaccination Point	Sanomala Vaccination Point
8	B13	Iso Omena Vaccination Point	Iso Omena Vaccination Point
9	B15	Malmi	Malmi
10	B16	Tapiola Health Center	Tapiola Health Center
11	B17	Myyrmäki Energia Areena	Myyrmäki Energia Areena
12	B18	Tapiola Health Center	Tapiola Health Center
13	B21	Iso Omena Vaccination Point	Iso Omena Vaccination Point
14	B22	Myyrmäki Energia Areena	Myyrmäki Energia Areena
15	B23	Sanomala Vaccination Point	Sanomala Vaccination Point
16	B24	Malmi	Malmi
17	B25	Malmi	Malmi
18	B27	Myyrmäki Energia Areena	Myyrmäki Energia Areena
19	B28	Iso Omena Vaccination Point	Iso Omena Vaccination Point
20	B29	Myyrmäki Energia Areena	Sanomala Vaccination Point
21	B30	Iso Omena Vaccination Point	Iso Omena Vaccination Point

	batchID	phone
0	B02	093-101-0024
1	B06	098-163-4500
2	B29	093-104-5930

Query 4

```
SELECT DISTINCT Patient.name AS patient, VaccinationEvent.batchID,
Vaccine.name AS vaccine, PatientVaccinationEvent.date,
PatientVaccinationEvent.location
FROM PatientVaccinationEvent
JOIN Diagnosis ON patientSsNo = patient
JOIN Symptom ON symptom = Symptom.name
JOIN Patient ON patient = ssNo
JOIN VaccinationEvent ON VaccinationEvent.date =
PatientVaccinationEvent.date AND PatientVaccinationEvent.location =
VaccinationEvent.location
JOIN VaccineBatch ON VaccinationEvent.batchID = VaccineBatch.batchID
JOIN Manufacturer ON Manufacturer.ID = manufacturerID
JOIN Vaccine ON vaccineID = Vaccine.ID
WHERE criticality = 1 AND "2021-05-10" < Diagnosis.date
ORDER BY patientSsNo, VaccinationEvent.date
```

Description

This query has been designed so that the table PatientVaccinationEvent is joined with the table Diagnosis based on the patient (the respective attributes are patientSsNo and patient), so that we can keep all the vaccination events for each patient while selecting only those with a diagnosis. Then, we proceed to join it with Symptom via the diagnosed symptoms in any patient (i.e. the attribute name from Symptom must match the attribute symptom in Diagnosis) to obtain whether they are critical (they are if the attribute criticality is 1) and then with the patient via social security number (ssNo in Patient and patient in Diagnosis) to obtain the name of the patient (attribute name in Patient). To obtain the batch of vaccines with which the patient was vaccinated, we need to join the VaccinationEvent table, taking into account that both the location and the date (same name of the attributes in both VaccinationEvent and PatientVaccinationEvent) matter. Finally, we find the vaccine name by joining Vaccine table via previous join of VaccineBatch and Manufacturer, since VaccineBatch only contains the ID of the manufacturer (and via VaccineBatch is the only way to obtain it), not which vaccine it is. To join those two tables to all the previous, we use the batchID (attribute with same name in VaccinationEvent and VaccineBatch), the manufacturer ID (manufacturerID in VaccineBatch and ID in Manufacturer), and the vaccine ID (vaccineID in Manufacturer and ID in Vaccine). Adding the condition for the date being after May 10, 2021, we can obtain as many tuples per patient as vaccination events he has received with the information about the vaccine received and the batch to which it belonged, the date and location where they happened, and the name of the patient. The patients are ordered by name and then by the date of their vaccination events so that it is visualized easily.

Result

No patient has reported any critical symptoms after May 10, 2021.

Query 5

```
CREATE VIEW VaccinationStatus AS
SELECT Patient.name Name, Patient.dateOfBirth DateOfBirth, Patient.gender
Gender, Patient.ssNo SocialSecurityNumber,
CASE WHEN Vaccine.doses = COUNT(PatientVaccinationEvent.patientSsNo) THEN
'1' ELSE '0' END AS vaccinationStatus
FROM Patient
LEFT JOIN PatientVaccinationEvent ON Patient.ssNo =
PatientVaccinationEvent.patientSsNo
LEFT JOIN VaccinationEvent ON (PatientVaccinationEvent.location =
VaccinationEvent.location) AND (PatientVaccinationEvent."date" =
VaccinationEvent."date")
LEFT JOIN VaccineBatch  ON VaccinationEvent.batchID = VaccineBatch.batchID
LEFT JOIN Manufacturer ON VaccineBatch.manufacturerID = Manufacturer.ID
LEFT JOIN Vaccine ON Manufacturer.vaccineID = Vaccine.ID
GROUP BY Patient.ssNo;
```

Description

There are numerous tables involved due to the distant relationship between the information about the vaccination event itself and the number of doses required. The From clause is populated with a series of Left Inner Joins among the tables Patient, PatientVaccinationEvent, VaccinationEvent, VaccineBatch, Manufacturer and Vaccine. This is to preserve the tuples that contain information on patients that have not been vaccinated yet. We then group the tuples by the social security number of each patient so that the table is easier to read, and finally, in the Select clause, we list all the attributes of the table Patient and use a Case When statement to create the table vaccinationStatus asked for in the problem statement. The condition used in the Case When statement is the number of vaccination events being equal to the required number of doses of the vaccine.

Result

As the question has only asked to create the view and there are 150 rows in the view the content of it is not shown here.

Query 6

```
SELECT location AS hospital, name AS vaccine, total AS "number of vaccines
of type", SUM(total) OVER (PARTITION BY location) AS "total number of
vaccines" FROM (
  SELECT location, Vaccine.name, SUM(amount) AS total FROM VaccineBatch
  JOIN Manufacturer ON Manufacturer.ID = VaccineBatch.manufacturerID
  JOIN Vaccine ON Vaccine.ID = Manufacturer.vaccineID
  GROUP BY location, vaccineID
)
```

Description

The data about the location of each vaccine batch is stored in the VaccineBatch table, so in this query the data is first read from that table and then joined with Manufacturer and Vaccine tables to get the name of the Vaccine in an inner query. Joining tables is required since in the VaccineBatch table only the manufacturer id is stored and the information on the type of the vaccine a manufacturer makes is stored in the Manufacturer table, and the last join with the Vaccine table is simply to get the name of the vaccine to show in the results. The data is grouped by location and vaccineID columns and the aggregation function SUM is used so the inner query returns the total number of each vaccine in each location.

The outer query is returning all the data that the inner query is returning, and on top of that it uses window aggregate functions to calculate the number of available vaccines in each location, the OVER (PARTITION BY) clause combined with SUM function results in the sum of the total column for each location. This adds the aggregated value as a separate column to every row of the result.

Result

	hospital	vaccine	number of vaccines of type	total number of vaccines
0	Iso Omena Vaccination Point	AstraZeneca	10	65
1	Iso Omena Vaccination Point	Moderna	30	65
2	Iso Omena Vaccination Point	Comirnaty	25	65
3	Malmi	AstraZeneca	20	65
4	Malmi	Moderna	30	65
5	Malmi	Comirnaty	15	65
6	Messukeskus	AstraZeneca	30	120
7	Messukeskus	Moderna	75	120
8	Messukeskus	Comirnaty	15	120
9	Myyrmäki Energia Areena	AstraZeneca	30	85
10	Myyrmäki Energia Areena	Moderna	30	85
11	Myyrmäki Energia Areena	Comirnaty	25	85
12	Sanomala Vaccination Point	AstraZeneca	10	40
13	Sanomala Vaccination Point	Moderna	30	40
14	Tapiola Health Center	AstraZeneca	10	55
15	Tapiola Health Center	Moderna	45	55

Query 7

```
WITH Raw AS
(
 SELECT PV.patientSsNo AS ssNo, V.name, symptom FROM
PatientVaccinationEvent PV
 JOIN VaccinationEvent VE ON VE.date = PV.date AND VE.location =
PV.location
 JOIN VaccineBatch VB ON VB.batchID = VE.batchID
 JOIN Manufacturer M ON M.ID = VB.manufacturerID
 JOIN Vaccine V ON V.ID = M.vaccineID
 JOIN Diagnosis D ON D.patient = PV.patientSsNo AND D.date > PV.date
),
Totals AS
(
 SELECT name, COUNT(DISTINCT(ssNo)) AS total FROM Raw GROUP BY name
```

```
),
ByType AS
(
 SELECT name, symptom, COUNT(DISTINCT(ssNo)) AS total FROM Raw GROUP BY
name, symptom
)
SELECT ByType.name AS vaccine, ByType.symptom, ByType.total * 1.0 /
Totals.total AS frequency FROM ByType JOIN Totals ON ByType.name =
Totals.name
```

Description

For this query the WITH clause is used to make calculating the frequency easier. The first sub-query block named Raw first joins the PatientVaccinationEvent table with VaccinationEvent table when date and location matches so that we know what vaccine batch is used. And then the data is joined with VaccineBatch, Manufacturer and Vaccine tables in order to find the name of the vaccine, similar to the query 6. The data that is required from this subquery is the social security number of the patient, name of the vaccine and the symptom.

In the Totals subquery, the data is grouped by the name of the vaccine and the number of patients who have received each vaccine is calculated using the COUNT(DISTINCT()) function.

In the ByType subquery, the data is grouped by the name of the vaccine and the symptom so the aggregation function returns the count of distinct patients who have received a vaccine and reported a specific symptom.

In the end the Total and ByType temporary tables are joined on the name of the vaccine and the frequency is calculated using the total columns of the two tables. Note that 1.0 is multiplied so the result is a floating-point number.

Result

	vaccine	symptom	frequency
0	AstraZeneca	blurring of vision	0.050000
1	AstraZeneca	diarrhea	0.050000
2	AstraZeneca	fatigue	0.050000
3	AstraZeneca	feelings of illness	0.050000
4	AstraZeneca	fever	0.150000
5	AstraZeneca	headache	0.300000
6	AstraZeneca	high fever	0.100000
7	AstraZeneca	inflammation near injection	0.050000
8	AstraZeneca	itchiness near injection	0.150000
9	AstraZeneca	joint pain	0.300000
10	AstraZeneca	muscle ache	0.350000
11	AstraZeneca	nausea	0.150000
12	AstraZeneca	vomiting	0.050000
13	AstraZeneca	warmth near injection	0.150000
14	Comirnaty	chest pain	0.083333
15	Comirnaty	diarrhea	0.166667
16	Comirnaty	fatigue	0.083333
17	Comirnaty	fever	0.250000
18	Comirnaty	headache	0.333333
19	Comirnaty	high fever	0.083333
20	Comirnaty	inflammation near injection	0.083333
21	Comirnaty	joint pain	0.166667
22	Comirnaty	muscle ache	0.250000
23	Comirnaty	pain near injection	0.083333
24	Moderna	chills	0.076923
25	Moderna	fatigue	0.076923
26	Moderna	feelings of illness	0.307692
27	Moderna	fever	0.153846
28	Moderna	headache	0.076923
29	Moderna	high fever	0.076923
30	Moderna	joint pain	0.307692
31	Moderna	lymfadenopathy	0.153846
32	Moderna	muscle ache	0.384615
33	Moderna	nausea	0.153846
34	Moderna	vomiting	0.076923