**Practicum – 1**

**Contact Tracing System for Epidemiologists and Epidemiological Research**

**Aman Batra**

**NUID: 001877232**

**Class: CS5200**

**Academic Term: Summer-2, 2020**

**Instructor: Dr. Martin Schedlbauer**

**Email :** [**Batra.am@northeastern.edu**](mailto:Batra.am@northeastern.edu)

**No Group – Individual Submission**

1. **Problem Definition – Contact Tracing**

**1.1 What is Contact Tracing?**

According to WHO: Contact tracing is the process of identifying, assessing, and managing people who have been exposed to a disease to prevent onward transmission.

**1.2 What is the purpose of Contact Tracing?**

When systematically applied, contact tracing will break the chains of transmission of COVID-19 to prevent future waves or surges of cases, and to enable us to get back to work in a much safer way. Contact Tracing is an essential public health tool for controlling the virus.

**1.3 How does Contact Tracing work?**

Contact tracing for COVID-19 requires identifying people who may have been exposed to COVID-19 and following them up daily for 14 days from the last point of exposure.

The goal is to create a spider web of coronavirus transmission

Multiple Approaches To Trace Contacts:

1. An application approach, Continuous subject monitoring and data gathering is achieved using a mobile-application. Patients use the application to self-assess symptoms and report their interactions with other contacts, which can then be notified via the app, and put in incubation. For Example, a portable contact tracing application with real-time threat notifications based on GPS location/Bluetooth tracking, daily self-assessments, and contact reporting.
2. A general approach, where infected person contacts local Public Health Authorities and notifies them, PHAs then takes report of the person via calls & interviews, and prescribes test or medical assessments. Daily reports and health updates are taken by the assigned personnel until the incubation period is active. All the contract tracing is done manually by the PHAs, who then feed the data into a central reporting system.
3. A combined approach, where both self-assessments and manual assessments are possible. Patients choose if they would like to contact and schedule meetings with PHAs or prefer the use of a real-time contact tracing app regularly. PHAs gather and anonymise all the data, and put it in a graph database. The graph databases are then analysed to create heatmaps of the COVID-19 affected areas, which are later turned into containment zones.
4. **Most Efficient Approach - Contact Tracing  
   (Combined Approach)**

Combined approach assumes that the local, regional, or national Public Health body of a geographical area, has offered three ways, to keep a check on COVID-19 epidemic: a mobile application, a website, and a dedicated COVID-19 helpline to implement contact tracing measures.

1. **Automated**: A self-assessment and self-reporting based portable application, capable of running on multiple types of devices such as mobile phones, tablets, and laptops with real-time geolocation and Bluetooth based proximity tracing of other users, running the same application of their devices. Each user’s ***HealthStatus*** tokens are broadcasted within a 30 metres radius, and every user of the application in the proximity range will receive these tokens. If a sick user is nearby, all the users in the vicinity will get a threat alert. The total interaction time or visit time of a user will be recorded whenever he or she, meets or passes by, a sick person, or visits a place. If a user is not feeling and suspects that he or she might have been exposed to COVID-19, then there is a self-assessment option which contains a predefined set of questions, that can predict the likelihood of infection. If the assessment score bypasses the defined threshold, all users in the vicinity are notified of potential threat; Local health authority(LHA) is notified via the application. The LHA then prescribes a suitable COVID-19, and if the user tests positive, an incubation period of 14 days is initiated, which contains daily self-assessments, self-quarantining, and self-reporting of every place visited, every person in touch, and every notable interaction made within the past 14 days.
2. **Semi-Automated:** Some users have privacy issues when it comes to using applications that continuously record user data, and keep surveillance over their activities. For such users, a website or a similar app without monitoring is a better option, to implement contact tracing. When such a user feels sick, he can go to the LHA website, and take a self-assessment test, if the test results indicate potential infection, their identity is anonymised and location data is fed into the central database(common to all strategies). The user has to take a daily self-assessment until he or she is marked ‘healthy’ again, and all the contact reporting is done on the website. The central database is used to create heatmaps of coronavirus stricken areas, the website users can manually check the heatmap zones online, whereas it is inbuilt in the application.
3. **Manual:** For users, who have no viable means to use the website or application, there is a dedicated helpline number, which takes care of daily assessments, and contact reporting. Things are done manually, via phone conversations or administered meetings, and all the gathered data is then manually uploaded to the database.
4. **Database Tools Specification**

**Overview**

The database for a Contact Tracing system is can be implemented using many relational and non-relational DBMS such MySQL, PostgreSQL, MongoDB, Neo4j, Oracle DB. Parts of the project can be implemented using graph-databases, because they will be best suited for running depth queries and discovering links at greater depths, but for this particular Practicum, we will be using MySQL as a DBMS tool ubiquitously, and R Studio for running Analytics. MySQL has many advantages and some limitations, as given below:

**Advantages of MySQL:**

1. Open source, inexpensive and readily available.
2. Industry Standard, and very popular.
3. Extensive support available online.
4. Ease, Intuitiveness and Usability
5. Outstanding InnoDB engine.

**Drawbacks**:

1. Scalability issues can arise with time.
2. Not very easy to debug.
3. Does not support very large databases efficiently.

**Advantages of R:**

1. Open source, Platform Independent
2. Rapid, and quality plotting
3. Non-Coder friendly, anyone can start plotting within a hours
4. Rich and continuously growing sets of packages (>10000) in the CRAN repository

**Drawbacks**:

1. R Utilizes more memory as objects are stored in the memory.
2. Slower than other programming languages like Python and Matlab
3. Does not support very large scale applications efficiently.
4. **Overall Assumptions & Constraints.**

**Assumptions:**

1. Only some parts of the whole application will be reflected in the relational database, some data will stored locally on the host devices, such as state variables, local variables, events data, device permissions details, etc.
2. There can be many use cases, but the database is designed keeping in mind only some of these use cases, hence some parts of the database can be missing For Example. The assessment-survey module can have 4 more classes, but we are only using one for now.
3. The depth of queries will be set to 5, because MySQL is not a graph-database, it takes a good amount of time and processing power to create Joins, and make connections.
4. Application logic and host application will be created at a later time.

**Constraints:**

* **Domain Level:**

1. **Varchar** for string data
2. **Integer** for whole numbers
3. **Boolean** for binary choices (True or False, Yes or No, Correct or Incorrect, these kind of choices will be implemented using 0s & 1s
4. **Enum** for Lists or categorical attributes
5. **Text** for descriptions

* **Referential Integrity:**
  1. *Place* extends to *Interactions, AppUser, Visits, PublicHealthAuthority*
  2. *Person* extends to *AppUser* and *PublicHealthWorker*
  3. *AppUser* extends to *Place, Interactions, Assessment, HealthReportCheck* and *Person*
  4. *UserEvents* extends to *Visits*, *Interactions* and *ContactHistoryLog*
  5. *PersonNotification, AppUserEmail, PersonPhone* are linking tables.
* **Entity Integrity**

1. All classes have primary keys, all primary keys are set to NOT NULL
2. Key constraints are enabled SET FOREIGN\_KEY\_CHECKS = 1
3. **UML Class Diagram**

**A close up of a map

Description automatically generated**

**Design Tool Used: Visual Paradigm Community Edition for Mac**

**Link to Files:**

1. **Entity Relationship Diagram**

Link to LucidChart: [https://app.lucidchart.com/documents/view/982d3b7d-538a-4993-9511-126f0853bb6e/0\_0#](https://app.lucidchart.com/documents/view/982d3b7d-538a-4993-9511-126f0853bb6e/0_0)

**A picture containing screenshot

Description automatically generated**

1. **Schema Generation &   
   Normalization Check Results**

**Relational Schema:**

Place(*placeint*,placename,streetarea,city,state,zip,latitude,longitude)

Person(*PersonID*,firstName,lastName)

PersonPhone(PhoneNo,PersonID)

AppUser(*UserID*,*AddressID*,RecoveredFromCovid)

AppUserEmail(EmailID, UserID)

PublicHealthWorker(WorkerID*,*title,*officeID*,PublicAuthID)

PublicHealthAuthority(*AuthorityID*,AuthAddressID,Name,Jurisdiction)

PublicHealthAuthorityHelpline(HelplineNo,AuthorityID)

UserEvent(*EventID*,EventStartTime,EventEndTime,Description)

Visits(VisitID,VisitingUserID,VisitPlaceID)

Interactions(InteractionID,interactingUserID,VisitorID,InteractionPlaceID)

Notifications(*Nid*,PHAAuthorityID,timestamp,OtherInformation)

HealthReportCheck(*ReportID*,UserID,Temperature,OxygenLevel,HealthStatus,TestResult,Descriptions,CheckingWorkerID, ReportDate)

Assessment(AssessID,TakerID,CovidSuspected,TimeStamp,ResponseSheet, AssesseeType, OverseeingWorkerID)

PersonNotification(Nid, PersonID)

**Normalization to BCNF**

This following table lists out every relation in the database and provides proof to make sure its in BCNF. There Is no need to prove lower normal forms like 1NF, 2NF and 3NF because, if a relationship in BCNF, it IMPLIES that it is already normalized in lower forms.

The relationships shows below, comply with all of the following criterion, needed for validating BCNF.

1. Every relationship has a valid candidate key as their determinants( All determinants are candidate keys)
2. There is no partial dependency of any kind
3. No composite candidate keys with overlapping attributes
4. No multivalued attributes exist
5. No transitive dependency.

A picture containing bird, flower

Description automatically generated A close up of a piece of paper

Description automatically generated

|  |  |  |
| --- | --- | --- |
| S.No. | Table Name | All Functional Dependencies |
| 1 | A screenshot of a cell phone  Description automatically generated  UserID=>addressid  UserID=>=>name  UserID=>birthdate | A close up of a sign  Description automatically generated   * **UserID** is a candidate key and Primary key which uniquely identifies Birthdate, AddressID, RecoveredFromCovid. * **AddressID** is a Candidate Key, because it can uniquely identify an AppUser * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 2 | A screenshot of a cell phone  Description automatically generated  EmailID -> UserId | EmailID -> UserId   * **UserID** is a candidate key and Primary key which uniquely identifies Birthdate, AddressID, RecoveredFromCovid. * **AddressID** is a Candidate Key, because it can uniquely identify an AppUser * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |

|  |  |  |
| --- | --- | --- |
| 3 | A screenshot of a cell phone  Description automatically generated  PersonID=>firstname  PersonID=>lastname | * **PersonID** is a candidate key and Primary key which uniquely identifies firstname, lastname * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 4 | A screenshot of a cell phone  Description automatically generated  PhoneNo=>PhoneID | A close up of a sign  Description automatically generated   * **PhoneNo** is a candidate key and Primary key which uniquely identifies PersonID * PersonID is not unique here because a user can have multiple phone numbers. * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 5 | A screenshot of a cell phone  Description automatically generated  AuthorityID=>AuthAddressID  AuthorityID=>Name  AuthorityID=>Jurisdiction | A close up of a sign  Description automatically generated   * **authorityid** is a candidate key and Primary key which uniquely identifies all other attributes * name, jurisdiction for a secondary relationship that can identify authorityid * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 6 | A screenshot of a cell phone  Description automatically generated  helplineno=>authoritid | * **helplineno** is a candidate key and Primary key which uniquely identifies AuthorityID, because a number cannot be shared by multiple authorities in our assumption * authorityID has multiple numbers hence not unique to relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |

|  |  |  |
| --- | --- | --- |
| 7 | A screenshot of a cell phone  Description automatically generated  Workerid => officeid  Workerid => Title  Workerid => publicauthid | A picture containing drawing  Description automatically generated   * **WorkerID** is a candidate key and Primary key which uniquely identifies Title, OfficeID, publicauthID * **OfficeID** is a candidate key which uniquely identified the PK * PubliAuthID has multiple workers so not unique to relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 8 | A screenshot of a cell phone  Description automatically generated  AssessId=>takerid  AssessId=>covidsuspected  AssessId=>timestamp  AssessId=>responsesheet  AssessId=>assesseetype  AssessId=>overseeingworkerid | A screenshot of a cell phone  Description automatically generated   * **WorkerID** is a candidate key and Primary key which uniquely identifies all the attributes * Same TakerID is on multiple assessments so not unique to relationship * Timestamp, takerid and overseeingworkeridform a secondary relationship (non-primes to key) * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |

|  |  |  |
| --- | --- | --- |
| 9 | A screenshot of a cell phone  Description automatically generated  ReportID=>userid  ReportID=>temperature  ReportID=>oxygenlevel  ReportID=>healthstatus  ReportID=>testresult  ReportID=>description  ReportID=>checkingworkerid  ReportID=>reportdate | * **ReportID** is a candidate key and Primary key which uniquely identifies all the attributes * Same UserID is on multiple Reports so not unique to relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 10 | A screenshot of a cell phone  Description automatically generated  InteractionID=>interactinguserid  InteractionID=>interactingplaceid  InteractionID=>visitid | * **InteractionID** is a candidate key and Primary key which uniquely identifies all the attributes * Same InteractingUseID, VisitorID and InteractionPlaceID can exist on multiple interactions so not unique to relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 11 | A screenshot of a cell phone  Description automatically generated  eventid=>logbookid  eventid=>description  eventid=>eventstarttime  eventid=>eventendtime | A close up of a sign  Description automatically generated   * **EventID** is a candidate key and Primary key which uniquely identifies logbookid, description, eventstarttime, eventendtime * Same set of start time, end time and logbookid can exist in case the database is really large so they are not unique to relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 12 | A screenshot of a cell phone  Description automatically generated  placeid=>placename  placeid=>streetarea  placeid=>state  placeid=>city  placeid=>coountry  placeid=>latitude  placeid=>longitude | * **placeid** is a candidate key and Primary key which uniquely identifies logbookid, description, eventstarttime, eventendtime * Same set of address and coordinates can exist in case for multiple persons sharing an address, so they are not unique to relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 13 | A screenshot of a cell phone  Description automatically generated | Nid,PersonID   * (Nid,PersonID) is the only candidate key and a primary key, no non-prime attribute exist, so no dependency * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 14 | A screenshot of a cell phone  Description automatically generated  nid=>PHAauthorityid  nid=>timestamp  nid=>otherinformation | A close up of a sign  Description automatically generated   * **nid** is a candidate key and Primary key which uniquely identifies PHAAuthorityID, timestamp, otherinformation * trivial attributes (phaauthorityid, timestamp, otherinformation) form a secondary relationship. (Non-prime to key) * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 15 | A screenshot of a cell phone  Description automatically generated  logid=>date  logid=>assignedworkerid | * **logid** is a candidate key and Primary key which uniquely identifies date and assignedworker * AssignedWorkerID can exist on multiple logs, so it is not unique to this relationship * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |
| 16 | A screenshot of a cell phone  Description automatically generated  visitid=>visitinguserid  visitid=>visitplaceid | * **VisitID** is a candidate key and Primary key which uniquely identifies VisitPlaceID and VisitingUserID * (VisitPlaceID,VisitingUserId) do not uniquely identify a visit because, there can be multiple visits, even on the same day * No Partial Dependency or Multivalued Attribute exist in this relationship * There is no non-trivial FD without a candidate key. Hence table is in BCNF |

1. **Integrity Checking Trials & Proofs**
2. ***Referential Integrity : Foreign Key Check***

**Test #1**

Attempt to deleted a referenced record:

DELETE from AppUser where userid = 1499;

Response:

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`contacttracingdb`.`appuseremail`, CONSTRAINT `appuseremail\_ibfk\_1` FOREIGN KEY (`UserID`) REFERENCES `appuser` (`UserID`)) 0.0039 sec

**Test#2**

Attempt to insert an unreferenced record:

INSERT INTO PublicHealthWorker VALUES(1600,'MisterA',131245,13112);

Response:

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`contacttracingdb`.`publichealthworker`, CONSTRAINT `publichealthworker\_ibfk\_2` FOREIGN KEY (`WorkerID`) REFERENCES `person` (`PersonID`))

1. ***Domain Integrity : Alien Value Check***

**Test #1**

Attempt to insert a non-allowed value:

INSERT INTO Assessment VALUES(3899,1600,0,'2020-05-29 08:37:55','a random description','Friend',1511)

Response:

Error Code: 1265. Data truncated for column 'AssesseeType' at row 1 0.00028 sec

Test#2

Attempt to INSERT a value outside bounds:

INSERT INTO Place VALUES(1002,'Parua','RODQ PLACE','EAST BOSTON','MA','2128',42.36443246,-731234568910);

Response:

Error Code: 1264. Out of range value for column 'longitude' at row 1 0.00026 sec

1. ***Entity Integrity and Key Constraints : Primary Key Null check***

Test #1

Attempt to INSERT a NULL value for PRIMARY KEY :

INSERT INTO AppUserEmail(UserID, EmailID) VALUES (1101,null);

Response:

Error Code: 1048. Column 'EmailID' cannot be null

Test#2

Attempt to INSERT a VALUE in AUTO\_INCREMENT Primary key field: We can clearly see that the Primary Key was autogenerated when a NULL value was passed.

INSERT INTO Place VALUES(NULL,'Aman','Batra PLACE','SOUTH BOSTON','MA','2128',42.36467840,-72.03322720);

**Select \* FROM PLACE where placename='Aman';**

Response:

1 row(s) affected.

**1001 Aman Batra PLACE SOUTH BOSTON MA 2128 42.36467840 -72.03322720**

Test#3: Checking Uniqueness of Keys

Counting primary keys of Places

SELECT placeid,COUNT(\*) as total FROM place GROUP BY placeid HAVING total > 1;

Response:

0 row(s) returned

1. **Create Tables & Insert Values**

**CREATE TABLE IF NOT EXISTS Place (**

**placeid INT NOT NULL AUTO\_INCREMENT,**

**placename VARCHAR(255) NOT NULL,**

**streetarea VARCHAR(255) NOT NULL,**

**state VARCHAR(255) NOT NULL,**

**city VARCHAR(255) NOT NULL,**

**zip VARCHAR(255) NOT NULL,**

**latitude decimal(10,8) signed NOT NULL,**

**longitude decimal(11,8) signed NOT NULL,**

**PRIMARY KEY (placeid)**

**);**

**CREATE TABLE IF NOT EXISTS Person (**

**PersonID INT NOT NULL AUTO\_INCREMENT,**

**firstName VARCHAR(255) NOT NULL,**

**lastName VARCHAR(255),**

**PRIMARY KEY (PersonID)**

**);**

**CREATE TABLE IF NOT EXISTS AppUser (**

**UserID INT NOT NULL,**

**Birthdate date NOT NULL,**

**AddressID INT NOT NULL,**

**RecoveredFromCovid Boolean,**

**PRIMARY KEY (UserID),**

**FOREIGN KEY(UserID) REFERENCES Person(PersonID),**

**FOREIGN KEY (AddressID) REFERENCES Place(PlaceID)**

**);**

**CREATE TABLE IF NOT EXISTS PublicHealthAuthority (**

**AuthorityID INT NOT NULL AUTO\_INCREMENT,**

**AuthAddressID INT NOT NULL,**

**Name VARCHAR(255) NOT NULL,**

**Jurisdiction VARCHAR(255),**

**PRIMARY KEY (AuthorityID),**

**FOREIGN KEY (AuthAddressID) REFERENCES Place(PlaceId)**

**);**

**CREATE TABLE IF NOT EXISTS PublicHealthWorker (**

**WorkerID INT NOT NULL,**

**Title VARCHAR(255) NOT NULL,**

**OfficeID VARCHAR(255) NOT NULL,**

**PublicAuthID INT NOT NULL,**

**PRIMARY KEY (WorkerID),**

**FOREIGN KEY (PublicAuthID) REFERENCES PublicHealthAuthority(AuthorityID),**

**FOREIGN KEY (WorkerID) REFERENCES Person(PersonId)**

**);**

**CREATE TABLE IF NOT EXISTS ContactHistoryLog (**

**LogID INT NOT NULL AUTO\_INCREMENT,**

**Date datetime,**

**AssignedWorkerID INT NOT NULL,**

**PRIMARY KEY (LogID),**

**FOREIGN KEY (AssignedWorkerID) REFERENCES PublicHealthWorker(WorkerId)**

**);**

**CREATE TABLE IF NOT EXISTS UserEvents (**

**EventID INT NOT NULL AUTO\_INCREMENT,**

**EventStartTime Datetime NOT NULL,**

**EventEndTime Datetime NOT NULL,**

**LogBookID INT NOT NULL,**

**Description VARCHAR(1200),**

**PRIMARY KEY (EventID),**

**FOREIGN KEY (LogBookID) REFERENCES ContactHistoryLog(LogID)**

**);**

**CREATE TABLE IF NOT EXISTS HealthReportCheck (**

**ReportID INT NOT NULL AUTO\_INCREMENT,**

**UserID INT NOT NULL,**

**Temperature decimal(10, 3),**

**OxygenLevel decimal(10, 3),**

**HealthStatus enum('sick', 'healthy') NOT NULL,**

**TestResult enum(**

**'Positive', 'Negative', 'Unclear'**

**) NOT NULL,**

**Description Text,**

**CheckingWorkerID INT NOT NULL,**

**ReportDate date,**

**PRIMARY KEY (ReportID),**

**FOREIGN KEY (UserID) REFERENCES AppUser(UserID),**

**FOREIGN KEY(CheckingWorkerID) REFERENCES PublicHealthWorker(WorkerId)**

**);**

**CREATE TABLE IF NOT EXISTS Assessment (**

**AssessID INT NOT NULL AUTO\_INCREMENT,**

**TakerID INT NOT NULL,**

**CovidSuspected boolean NOT NULL,**

**TimeStamp Datetime NOT NULL,**

**ResponseSheet VARCHAR(255),**

**AssesseeType enum('self', 'doctor', 'others') NOT NULL,**

**OverseeingWorkerID INT NOT NULL,**

**PRIMARY KEY (AssessID),**

**FOREIGN KEY (TakerID) REFERENCES AppUser(UserId),**

**FOREIGN KEY (OverseeingWorkerID) REFERENCES PublicHealthWorker(WorkerId)**

**);**

**CREATE TABLE IF NOT EXISTS AppUserEmail (**

**UserID INT NOT NULL,**

**EmailID VARCHAR(255) NOT NULL,**

**PRIMARY KEY (EmailID),**

**FOREIGN KEY(UserID) REFERENCES AppUser(UserID)**

**);**

**CREATE TABLE IF NOT EXISTS PublicHealthAuthorityHelpline(**

**HelplineNo VARCHAR(255) NOT NULL,**

**AuthorityID INT NOT NULL,**

**PRIMARY KEY(HelplineNo),**

**FOREIGN KEY(AuthorityID) REFERENCES PublicHealthAuthority(AuthorityID)**

**);**

**CREATE TABLE IF NOT EXISTS Interactions (**

**InteractionID INT NOT NULL,**

**InteractingUserID INT NOT NULL,**

**VisitorID INT NOT NULL,**

**InteractionPlaceID INT NOT NULL,**

**PRIMARY KEY (InteractionID),**

**FOREIGN KEY (InteractionID) REFERENCES UserEvents(EventID),**

**FOREIGN KEY (InteractingUserID) REFERENCES AppUser(UserID),**

**FOREIGN KEY (VisitorID) REFERENCES Person(PersonID),**

**FOREIGN KEY (InteractionPlaceID) REFERENCES Place(PlaceId)**

**);**

**CREATE TABLE IF NOT EXISTS Visit (**

**VisitID INT NOT NULL,**

**VisitPlaceID INT NOT NULL,**

**VisitingUserID INT NOT NULL,**

**PRIMARY KEY (VisitID),**

**FOREIGN KEY (VisitID) REFERENCES UserEvents(EventID),**

**FOREIGN KEY (VisitPlaceID) REFERENCES Place(PlaceID),**

**FOREIGN KEY(VisitingUserID) REFERENCES AppUser(UserID)**

**);**

**CREATE TABLE IF NOT EXISTS Notifications (**

**Nid INT NOT NULL AUTO\_INCREMENT,**

**PHAAuthorityID INT NOT NULL,**

**timeStamp datetime NOT NULL,**

**OtherInformation Text,**

**PRIMARY KEY (Nid),**

**FOREIGN KEY (PHAAuthorityID) REFERENCES PublicHealthAuthority(AuthorityId)**

**);**

**CREATE TABLE IF NOT EXISTS PersonNotification (**

**Nid INT NOT NULL,**

**PersonID INT NOT NULL,**

**PRIMARY KEY (Nid, PersonID),**

**FOREIGN KEY(Nid) REFERENCES Notifications(Nid),**

**FOREIGN KEY(PersonID) REFERENCES Person(PersonID)**

**);**

**CREATE TABLE IF NOT EXISTS PersonPhone(**

**PersonID INT NOT NULL,**

**PhoneNo VARCHAR(20) NOT NULL,**

**PRIMARY KEY (PhoneNo),**

**FOREIGN KEY (PersonID) REFERENCES Person(PersonID)**

**);**

**A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

1. **Query Generation, Scripts & Outputs**

QUERY – 1 – A SIMPLE JOIN QUERY

Joining Tables: AppUser, Person, AppUserEmail

Goal : To find all the details of a user who tested positive in COVID-19 Drug Test

SQL:

SELECT \* FROM AppUser

INNER JOIN Person ON AppUser.UserID=Person.PersonID

INNER JOIN AppUserEmail on AppUser.UserID=AppUserEmail.UserID

INNER JOIN Place on AppUser.AddressID=Place.placeid

INNER JOIN HealthReportCheck on AppUser.UserId=HealthReportCheck.UserID

WHERE HealthReportCheck.TestResult='positive';

Response: 153 row(s) returned

Result Screenshots:

* Divided into two parts

PART 1

A screenshot of a computer

Description automatically generated

PART 2

A screen shot of a computer

Description automatically generated

QUERY 2 – A Subquery to count total number of possible cases of direct person to person transmission in Massachusetts state.

SQL:

SELECT count(\*) as "Total probable cases of Direct Transmission via person to person interactions in Massachusetts"

FROM

(

SELECT i.InteractingUserID FROM Interactions i

INNER JOIN UserEvents on i.InteractionID=UserEvents.EventID

INNER JOIN HealthReportCheck r1 on i.InteractingUserID=r1.UserID

INNER JOIN HealthReportCheck r2 on i.VisitorID=r2.UserID

INNER JOIN Place on i.InteractionPlaceID=place.placeid

WHERE r1.TestResult='POSITIVE' and r2.TestResult in ('NEGATIVE','UNCLEAR')

AND r1.ReportDate<r2.ReportDate

AND Place.State in ('MA')

) as derived;

1 row(s) returned

Result Screenshots:

A screenshot of a cell phone

Description automatically generated

QUERY 3 – A QUERY with a HAVING CLAUSE to return the details of visits made by people who tested positive, within 20 days(before and after) of getting the test reports. Where the visit lasted longer than 4 hours.

SQL:

SELECT a.UserID, p.firstName, s.PhoneNo, u.EventStartTime as "Time of Visit",TIMEDIFF(u.EventEndTime,u.EventStartTime)as DurationOfVisit, m.placename,m.streetarea, m.city, m.longitude, m.latitude FROM visit v

INNER JOIN AppUser a on v.VisitingUserID=a.UserID

INNER JOIN Person p on a.userid=p.PersonID

INNER JOIN PersonPhone s on p.PersonID=s.PersonID

INNER JOIN Place m on v.VisitPlaceID=m.placeid

INNER JOIN UserEvents u on v.VisitID=u.EventID

INNER JOIN HealthReportCheck h on a.UserID=h.UserID

WHERE h.TestResult='POSITIVE'

AND datediff(h.ReportDate, u.EventStartTime)<20

AND datediff(h.ReportDate, u.EventStartTime)>-20

HAVING DurationOfVisit>'04:00:00'

ORDER by DurationOfVisit

9 row(s) returned

RESULT SCREENSHOTS:

A picture containing sitting, screen

Description automatically generated

QUERY 4 – A COMPLEX QUERY – A depth-2 search query to find out all interactions of persons, who came out positive/unclear in the covid-19 test, after their interacting with some person in the past.

Example Case:

* John Doe meets Sabrina Chan, and Jian Yang.
* John Doe was found positive, so Sabrina Chan and Jian Yang take covid tests.
* Jian tests positive, while Sabrina tests negative, so we need to trace Jian’s interactions now.

SQL:

select o.FirstName as "Guy1 who found out he got exposed", p.FirstName as "Guy2 who had met guy1", s.PhoneNo as "Phone number of Guy2", u.eventstarttime as "Interaction Start Details" , u.eventendtime as "Interaction End Details" from interactions a, person p, personphone s,userevents u, person o

WHERE a.InteractionID=u.eventid

AND p.personid = a.visitorid

AND s.PersonID=p.PersonId

AND a.Interactinguserid=o.personid

AND a.interactinguserid IN (

SELECT i.VisitorID FROM Interactions i

INNER JOIN UserEvents on i.InteractionID=UserEvents.EventID

INNER JOIN HealthReportCheck r1 on i.InteractingUserID=r1.UserID

INNER JOIN HealthReportCheck r2 on i.VisitorID=r2.UserID

WHERE r1.TestResult='POSITIVE' and r2.TestResult in ('UNCLEAR','POSITIVE')

AND r1.ReportDate< r2.ReportDate

)

AND a.Interactinguserid NOT IN (

SELECT i.interactinguserid FROM Interactions i

INNER JOIN UserEvents on i.InteractionID=UserEvents.EventID

INNER JOIN HealthReportCheck r1 on i.InteractingUserID=r1.UserID

INNER JOIN HealthReportCheck r2 on i.VisitorID=r2.UserID

WHERE r1.TestResult='POSITIVE' and r2.TestResult in ('UNCLEAR','POSITIVE')

AND r1.ReportDate< r2.ReportDate

)

37 row(s) returned

RESULT SCREENSHOTS

A picture containing computer

Description automatically generated

Note: We don’t use the distinct keyword because we want to find out all interactions, even if there were more than one!

We can go till depth 5 with MySQL in a medium sized database, after depth-5 mysqld crashes within 30 minutes of wait.

QUERY 5 – A Query of Choice

Details of interactions of persons who tested COVID positive(Depth 1 -> Guy 0 meets Guy 1)

SQL:

SELECT i.interactingUserId as UniqueID, guy1.firstName as "Meeting Person 1", r1.TestResult as "First Guy's Covid Report"

,i.VisitorID as UniqueID,guy2.firstName as "Meeting Person 2", r2.TestResult as "Second Guy's Covid Report", UserEvents.EventStartTime as "DateTime of Meeting", UserEvents.Description as "Details of Meeting"

FROM Interactions i

INNER JOIN UserEvents on i.InteractionID=UserEvents.EventID

INNER JOIN Person guy1 on i.InteractingUserID=guy1.PersonID

INNER JOIN Person guy2 on i.VisitorID=guy2.PersonID

INNER JOIN HealthReportCheck r1 on i.InteractingUserID=r1.UserID

INNER JOIN HealthReportCheck r2 on i.VisitorID=r2.UserID

WHERE r1.TestResult='POSITIVE'

228 row(s) returned

RESULTS SCREENSHOTS:

A close up of a computer

Description automatically generated

QUERY 6 – EXTRA QUERY

A QUERY with HAVING clause to count the no of people per city above 60 years in age, who took the Covid-19 Assessment survey, and suspected an infection, Only listing cities with more than 10 distinct suspects.

SQL:

SELECT distinct Place.city, count(distinct Assessment.TakerID) as "Total Covid Suspects around Boston region, with age greater than 50" FROM Place

INNER JOIN AppUser on Place.placeid=AppUser.addressid

INNER JOIN Assessment on AppUser.UserID=Assessment.TakerID

WHERE Assessment.CovidSuspected=1 AND AppUser.Birthdate<'1960-01-01'

Group By Place.City HAVING count(distinct Assessment.TakerID)>10;

Result Screenshots:

A screenshot of a cell phone

Description automatically generated

QUERY 7 – EXTRA QUERY  
Exposure events and places and Massachusetts. This query backtracks the visit log of a COVID-19 positive persons and returns exact places where they visited and spent some time.

SELECT m.placename,streetarea, m.city,u.EventStartTime as ExposureStart, u.EventEndTime as ExposureEnd FROM visit v

INNER JOIN AppUser a on v.VisitingUserID=a.UserID

INNER JOIN Person p on a.userid=p.PersonID

INNER JOIN PersonPhone s on p.PersonID=s.PersonID

INNER JOIN Place m on v.VisitPlaceID=m.placeid

INNER JOIN UserEvents u on v.VisitID=u.EventID

INNER JOIN HealthReportCheck h on a.UserID=h.UserID

WHERE h.TestResult='POSITIVE'

AND M.state='MA'

AND datediff(h.ReportDate, u.EventStartTime)<20

AND datediff(h.ReportDate, u.EventStartTime)>-20

10 row(s) returned

RESULT SCREENSHOTS:

A picture containing text, screen, sitting, large

Description automatically generated

1. **MySQL Integration with R-Studio, Integrated Queries and Plots**