**POTHOLE TRACKER AND REPAIR MANAGEMENT SYSTEM**



*Image Reference: https://www.india.com/*

# Contents:

1. Organization Description
2. Entity Relationship Model
3. Logical Schema Design
4. Data Dictionary
5. Create Table Queries
6. Triggers
7. Views

# Organization Description

**“More deadly than terrorism: Potholes responsible for killing 10 people a day in India” – guardian.com (Jul-25-2018)**

Roads with potholes has become a ubiquitous problem in India. According to a leading daily newspaper, in 2018(till July) alone 9300 people were killed and nearly 25000 were injured in road accidents which were caused due to potholes. This motivated me to create a mobile application known as Pothole Stalker which will help the government in recognizing these potholes and carry out their repair as soon as possible to avoid the inconveniences caused to the public. This mobile application will use mobile in-build gyroscope and Global Positioning System(GPS) to track the movement of the vehicle and if in case the vehicle crosses a pothole, it will trigger an API call with relevant information about the pothole.

The purpose of this non – profit business organization is to help the Indian Government and also alert the user of this application so as to accidents that happen due to these potholes can be avoided.

For this application to work efficiently, database is of utmost importance. Pothole Tracker and Repair Management System will ensure that the purpose of the application is fulfilled.

Anybody can download the application. The database will record information about the User. If the user feels a bump, the data about the pothole such as latitude, longitude, depth, RoadID and timestamp will be recorded in a Pothole-Event table. One user can provide many pothole events. The pothole events will then be summarized as Min latitude, Max latitude, Min longitude, Max longitude to find the pothole region. The PotholeID will connect it to the road where the pothole region is identified. Data such as SpeedLimit permissible, date of construction completed, traffic density will be recorded.

India has the second largest road network in the world, these roads can be divided into three categories:

1. National Highways: that connect one state to another
2. State Highways: that connect cities within a state
3. Urban Roads: that are used for travelling within a city.

These roads come under different jurisdiction. For example Central government is responsible for construction and maintenance of National Highways (National Highway Authority of India) and State Highways and Urban Roads construction and maintenance is the responsibility of the State government(State Public Works Department). Also in India, roads are constructed and maintained with the help of the public private partnership. After recognizing the pothole region and the type of road, the respective jurisdiction with the help of the Contractor will start a repair work and the status of the repair will also be recorded in the repair table. The Contractor who built the road will provide repair workers and Quality Inspector allocated by the jurisdiction will oversee the repair work. Each Quality Inspector will supervise many repair projects. The application will also track User’s latitude and longitude every minute of the drive which will be recorded in the Location Event table. If the distance between the pothole region and user is 1km, an alert will be recorded in the alert table.

**Values Gained by Database Development**

1. Main purpose of this Pothole Tracking and Repair Management is to save as many lives as possible as this system will provide assistance in speeding up the repair work of the pothole.
2. Corruption and incompetence of various parties involved in road construction and management can also be kept in check.
3. Citizens can contribute and help the government in keeping track of the road condition and its maintenance

# Entity – Relationship Diagram

# Create Table Queries

1. User

CREATE TABLE User\_T   
  (   
     UserID                    *bigint* NOT NULL CHECK(UserID > 0),   
     UserFirstName             *nvarchar*(100) NOT NULL,   
     UserLastName              *nvarchar*(100) NOT NULL,   
     UserDateOfBirth           *date*,   
     UserMobileNumber          *char*(10) NOT NULL,   
     UserEmail                 *nvarchar*(200),   
     VehicleRegistrationNumber *varchar*(20),   
     Password                  *nvarchar*(100) NOT NULL,   
     CONSTRAINT User\_PK PRIMARY KEY (UserID))

1. AddressPincode

As address was used in many tables, the city and state are transitively dependent on pincode.So I have created this table to full the requirements of 3NF

CREATE TABLE AddressPincode\_T (   
     PincodeID *bigint* not null check(PincodeID > 0),   
     Pincode   *int*,   
     City      *nvarchar*(100),   
     StateName *nvarchar*(100),   
     Constraint AddressPincode\_PK primary key(PincodeID))

1. Contractor

This table stores the contractor details and PincodeID is used as a foreign key to store the contractor’s address

CREATE TABLE Contractor\_T (   
     ContractorID          *bigint* NOT NULL CHECK(ContractorID > 0),   
     ContractorName        *nvarchar*(100) NOT NULL,   
     ContractorPhoneNumber *char*(10),   
     ContractorStreet      *nvarchar*(100),   
     PincodeID             *bigint* NOT NULL CHECK(PincodeID > 0),   
     CONSTRAINT Contractor\_PK PRIMARY KEY(ContractorID),   
     CONSTRAINT AddressPincode\_FK FOREIGN KEY(PincodeID) REFERENCES   
     AddressPincode\_T(PincodeID))

1. Road

This table stores the details about road and ContractorID is used as a foreign key to know which contractor built this road.

CREATE TABLE Road\_T (   
     RoadID                      *bigint* NOT NULL CHECK(RoadID > 0),   
     RoadName                    *nvarchar*(100) NOT NULL,   
     RoadLength                  *bigint*,   
     SpeedLimit                  *int*,   
     DateOfConstructionCompleted *date*,   
     TrafficDensity              *int*,   
     ContractorID                *bigint* NOT NULL CHECK(ContractorID > 0),   
     TypeOfRoad                  *varchar*(3) CHECK(TypeOfRoad IN ('N', 'S', 'U'))   
     NOT NULL,   
     CONSTRAINT Road\_PK PRIMARY KEY(RoadID),   
     CONSTRAINT Contractor\_FK FOREIGN KEY(ContractorID) REFERENCES Contractor\_T(   
     ContractorID))

1. Jurisdiction

This table stores the details of the person to be contacted in jurisdidction for pothole repair and informs about the type of jurisdiction

CREATE TABLE Jurisdiction\_T (   
     JurisdictionID     *bigint* NOT NULL CHECK(JurisdictionID > 0),   
     JContactPersonName *nvarchar*(100),   
     JPhoneNumber       *char*(10),   
     TypeOfJurisdiction *varchar*(2) CHECK(TypeOfJurisdiction IN('St', 'Ct')) NOT   
     NULL,   
     CONSTRAINT Jurisdiction\_PK PRIMARY KEY(JurisdictionID))

1. StateGovernment

This table stores details about the state government who will be responsible for state highways and urban roads. As it is a subtype of jurisdiction,its primary key will be used as a foreign key

CREATE TABLE StateGovernment\_T (   
     StJurisdictionID *bigint* NOT NULL CHECK(StJurisdictionID > 0),   
     PWDName          *nvarchar*(100),   
     PWDStreetName    *nvarchar*(100),   
     PincodeID        *bigint* NOT NULL CHECK(PincodeID > 0),   
     CONSTRAINT StateGovernment\_PK PRIMARY KEY(StJurisdictionID),   
     CONSTRAINT StateGovernment\_FK1 FOREIGN KEY(StJurisdictionID) REFERENCES   
     Jurisdiction\_T(JurisdictionID),   
     CONSTRAINT AddressPincode\_FK2 FOREIGN KEY(PincodeID) REFERENCES   
     AddressPincode\_T(PincodeID))

1. CentralGovernment

This table stores details about the central government who will be responsible for National highways As it is a subtype of jurisdiction, its primary key will be used as a foreign key

CREATE TABLE CentralGovernment\_T (   
     CtJurisdictionID   *bigint* NOT NULL CHECK(CtJurisdictionID > 0),   
     NHAIRouteNumber    *int*,   
     NHAIRegionalOffice *nvarchar*(100),   
     NHAIStreetName     *nvarchar*(100),   
     PincodeID          *bigint* NOT NULL CHECK(PincodeID > 0),   
     CONSTRAINT CentralGovernment\_PK PRIMARY KEY(CtJurisdictionID),   
     CONSTRAINT CentralGovernment\_FK1 FOREIGN KEY(CtJurisdictionID) REFERENCES   
     Jurisdiction\_T(JurisdictionID),   
     CONSTRAINT AddressPincode\_FK3 FOREIGN KEY(PincodeID) REFERENCES   
     AddressPincode\_T(PincodeID))

1. StateHighways

This table stores details about the cities connected by state highways and it is a subtype of road its primary key is also its foreign key and foreign key StJurisdictionID informs which jurisdiction it comes under

CREATE TABLE StateHighways\_T  (   
     SRoadID          *bigint* NOT NULL CHECK(SRoadID > 0),   
     FromCity         *nvarchar*(100),   
     ToCity           *nvarchar*(100),   
     StJurisdictionID *bigint* NOT NULL CHECK(StJurisdictionID > 0),   
     CONSTRAINT StateHighways\_PK PRIMARY KEY(SRoadID),   
     CONSTRAINT StateHighways\_FK1 FOREIGN KEY(SRoadID) REFERENCES Road\_T(RoadID)   
     ,   
     CONSTRAINT StateGovernment\_FK2 FOREIGN KEY(StJurisdictionID) REFERENCES   
     StateGovernment\_T(StJurisdictionID)   
  )

1. NationalHighways

This table stores details about the state capital onnected by National highways and it is a subtype of road its primary key is also its foreign key and foreign key CtJurisdictionID informs which jurisdiction it comes under

CREATE TABLE NationalHighways\_T (   
     NRoadID                *bigint* NOT NULL CHECK(NRoadID > 0),   
     FromStateCapital       *nvarchar*(100),   
     FromState              *nvarchar*(100),   
     ToStateCapital         *nvarchar*(100),   
     ToState                *nvarchar*(100),   
     NumberOfStatesCrossing *int*,   
     CtJurisdictionID       *bigint* NOT NULL CHECK(CtJurisdictionID > 0)   
     CONSTRAINT NationalHighways\_PK PRIMARY KEY(NRoadID),   
     CONSTRAINT NationalHighways\_FK1 FOREIGN KEY(NRoadID) REFERENCES Road\_T(   
     RoadID),   
     CONSTRAINT CentralGovernment\_FK2 FOREIGN KEY(CtJurisdictionID) REFERENCES   
     CentralGovernment\_T(CtJurisdictionID))

1. UrbanRoads

This table stores details about the district in the city where the road is and it is a subtype of road its primary key is also its foreign key and foreign key StJurisdictionID informs which jurisdiction it comes under

CREATE TABLE UrbanRoads\_T (   
     URoadID          *bigint* NOT NULL CHECK(URoadID > 0),   
     UDistrictName    *nvarchar*(100),   
     PincodeID        *bigint* NOT NULL CHECK(PincodeID > 0),   
     StJurisdictionID *bigint* NOT NULL CHECK(StJurisdictionID > 0),   
     CONSTRAINT UrbanRoads\_PK PRIMARY KEY(URoadID),   
     CONSTRAINT UrbanRoads\_FK1 FOREIGN KEY(URoadID) REFERENCES Road\_T(RoadID),   
     CONSTRAINT StateGovernment\_FK3 FOREIGN KEY(StJurisdictionID) REFERENCES   
     StateGovernment\_T(StJurisdictionID),   
     CONSTRAINT AddressPincode\_FK6 FOREIGN KEY(PincodeID) REFERENCES   
     AddressPincode\_T(PincodeID))

1. PotholeEvent

This table stores details about the Pothole events sent by the user. Foreign key RoadID informs about which road it is on and foreign key informs about the user that provided it.

CREATE TABLE PotholeEvent\_T (   
     PotholeEventID *bigint* CHECK(PotholeEventID > 0),   
     PLatitude      *decimal*(10, 8),   
     PLongitude     *decimal*(11, 8),   
     PotholeDepth   *float*,   
     Timestamp      *datetime*,   
     RoadID         *bigint* CHECK(RoadID > 0),   
     PotholeID      *bigint*,   
     UserID         *bigint* CHECK(UserID > 0),   
     CONSTRAINT PotholeEvent\_PK PRIMARY KEY(PotholeID),   
     CONSTRAINT Road\_FK1 FOREIGN KEY(RoadID) REFERENCES Road\_T(RoadID),   
     CONSTRAINT Pothole\_FK2 FOREIGN KEY(PotholeID) REFERENCES PotholeSummary\_T,   
     CONSTRAINT User\_FK2 FOREIGN KEY(UserID) REFERENCES User\_T(UserID))

1. PotholeSummary

This table summarizes the pothole event to find the pothole region and RoadID (foreign key) informs about the road where the pothole is

CREATE TABLE PotholeSummary\_T (   
     PotholeID          *bigint* NOT NULL IDENTITY(1, 1),   
     MinLatitude        *decimal*(10, 8),   
     MaxLatitude        *decimal*(10, 8),   
     MinLongitude       *decimal*(11, 8),   
     MaxLongitude       *decimal*(11, 8),   
     AvgPotholeDepth    *float*,   
     VehiclesCrossed    *bigint*,   
     TotalDepthRecorded *float*,   
     LatestTimestamp    *datetime*,   
     RoadID             *bigint* CHECK(RoadID > 0),   
     CONSTRAINT PotholeSummary\_PK PRIMARY KEY (PotholeID),   
     CONSTRAINT Road\_FK FOREIGN KEY (RoadID) REFERENCES Road\_T(RoadID))

1. StateHighways

This table stores details about quality inspector who oversee the repair work and JurisdictionID (foreign key) tells jurisdiction allotted him/her

CREATE TABLE QualityInspector\_T (   
     QInspectorID           *bigint* NOT NULL CHECK(QInspectorID > 0),   
     QFirstName             *nvarchar*(100),   
     QLastname              *nvarchar*(100),   
     QPhoneNumber           *char*(10),   
     NumberOfRepairProjects *int*,   
     JurisdictionID         *bigint* NOT NULL CHECK(JurisdictionID > 0)   
     CONSTRAINT QualityInspector\_PK PRIMARY KEY (QInspectorID),   
     CONSTRAINT Jurisdiction\_FK FOREIGN KEY (JurisdictionID) REFERENCES   
     Jurisdiction\_T(JurisdictionID))

1. Status

This table stores repair work status.

CREATE TABLE Status\_T (   
     StatusID          *bigint* NOT NULL CHECK(StatusID > 0),   
     StatusDescription *nvarchar*(100),   
     CONSTRAINT Status\_PK PRIMARY KEY (StatusID))

1. RepairWork

This table stores details about the pothole repair work and foreign key StatusID informs the status, PotholeID tells which on pothole repair work is going on and QualityInpectorID tells which Quality Inspector is overseeing its work

CREATE TABLE RepairWork\_T (   
     RepairID        *bigint* NOT NULL CHECK(RepairID > 0),   
     RepairStartDate *date*,   
     NumberOfWorkers *int*,   
     StatusID        *bigint* NOT NULL CHECK(StatusID > 0),   
     PotholeID       *bigint* NOT NULL CHECK(PotholeID > 0),   
     QInspectorID    *bigint* NOT NULL CHECK(QInspectorID > 0),   
     CONSTRAINT RepairWork\_PK PRIMARY KEY(RepairID),   
     CONSTRAINT Status\_FK1 FOREIGN KEY(StatusID) REFERENCES Status\_T(StatusID),   
     CONSTRAINT PotholeSummary\_FK2 FOREIGN KEY(PotholeID) REFERENCES   
     PotholeSummary\_T(PotholeID),   
     CONSTRAINT QualityInspector\_FK3 FOREIGN KEY(QInspectorID) REFERENCES   
     QualityInspector\_T(QInspectorID))

1. RepairWorker

This table stores details about the pothole repair workers. Foreign key repair id informs about the repair work, contractor id tells repair workers are from which contractor and pincode id is used for storing there address.

CREATE TABLE RepairWorkers\_T (   
     WorkerID          *bigint* NOT NULL CHECK(WorkerID > 0),   
     WorkerFirstName   *nvarchar*(100),   
     WorkerLastName    *nvarchar*(100),   
     WorkerStreetName  *nvarchar*(100),   
     PincodeID         *bigint* NOT NULL CHECK(PincodeID > 0),   
     WorkerPhoneNumber *char*(10),   
     ContractorID      *bigint* NOT NULL CHECK(ContractorID > 0),   
     RepairID          *bigint* CHECK(RepairId > 0),   
     CONSTRAINT RepairWorkers\_PK PRIMARY KEY(WorkerID),   
     CONSTRAINT Contractor\_FK1 FOREIGN KEY(ContractorID) REFERENCES Contractor\_T   
     (ContractorId),   
     CONSTRAINT RepairWork\_FK2 FOREIGN KEY(RepairID) REFERENCES RepairWork\_T(   
     RepairID),   
     CONSTRAINT AddressPincode\_FK3 FOREIGN KEY(PincodeID) REFERENCES   
     AddressPincode\_T(PincodeID))

1. LocationEvent

When user is driving their mobile application is suppose to share location. This table collects location, road information (road id) and user information(user\_id) coming as the foreign key.

CREATE TABLE LocationEvent\_T (   
     LocationEventID *bigint* CHECK(LocationEventID > 0),   
     LocLatitude *decimal*(10, 8),LocLongitude *decimal*(11, 8),   
     LocTimeStamp *datetime*,RoadID *bigint* CHECK(RoadID > 0),   
     UserID *bigint* CHECK(UserID > 0),   
     CONSTRAINT LocationEvent\_PK PRIMARY KEY(LocationEventID),   
     CONSTRAINT Road\_FK1 FOREIGN KEY(RoadID) REFERENCES Road\_T(RoadID),   
     CONSTRAINT User\_FK2 FOREIGN KEY(UserID) REFERENCES User\_T(UserID)   
  )

18. Pothole Alert

If user is close to the pothole, trigger will automatically run on every entry of location event table, if user is close to the pothole then an alert entry is made to this table. This table contains user information, location information and pothole information as the foreign keys.

CREATE TABLE PotholeAlert\_T (   
     PotholeAlertID *bigint* NOT NULL CHECK(AlertID > 0),   
     PotholeAlertMessage *nvarchar*(200),UserID *bigint* NOT NULL CHECK(UserID > 0),   
     LocationEventID *not* NULL CHECK(LocationEventID > 0),   
     CONSTRAINT PotholeAlert\_PK PRIMARY KEY(AlertID),   
     CONSTRAINT User\_FK1 FOREIGN KEY(UserID) REFERENCES User\_T(UserID)   
     CONSTRAINT LocationEvent\_FK2 FOREIGN KEY(LocationEventID) REFERENCES   
     LocationEvent\_T(LocationEventID)   
     CONSTRAINT PotholeSummary\_FK3 FOREIGN KEY(PotholeID) REFERENCES   
     PotholeSummary\_T(PotholeID)   
  )

# Trigger

## Trigger – Pothole Event

On every row insertion the TRIGGER will look whether this pothole event is close to the values already present in the pothole summary table. For that I am using HAVERSINE formula to compare lat/long for the POTHOLE\_EVENT with min/max lat/long of all the unique pothole entries in the summary table :

IF this lat/long is closer to the boundaries then the trigger will update the **PotholeSummary** table as it will find that pothole already exist.

ELSE trigger will take this as a new pothole and insert it as a new row in the summary table.

In the end PotholeID from **PotholeSummary** Table gets updated to **PotholeEventTable** for relationship mapping.

CREATE TRIGGER [dbo].[trigger1]   
ON [dbo].[PotholeEvent\_T]   
after INSERT   
AS   
    DECLARE @pLat *DECIMAL*(10, 8), @pLong *DECIMAL*(11, 8),  @pDepth *FLOAT*, @pTimestamp *DATETIME*,@roadId *BIGINT*, @userId *BIGINT*,@pEventId *BIGINT*, @minLat *DECIMAL*(10, 8), @maxLat *DECIMAL*(10, 8),@minLong *DECIMAL*(11, 8), @maxLong *DECIMAL*(11, 8),@avgDepth           *FLOAT*, @totalDepthRecorded *FLOAT*, @potholeId *BIGINT*, @vehicleCrossed *BIGINT*, @latestTimestamp *DATETIME*   
  
    SELECT @pLat = a.PLatitude,@pLong = a.PLongitude,@pDepth = a.PotholeDepth, @roadId = a.RoadID,@userId = a.UserID,@pTimestamp = a.Timestamp,@pEventId = a.PotholeEventID,@minLat = b.MinLatitude,@maxLat = b.MaxLatitude,@minLong = b.MinLongitude,@maxLong = b.MaxLongitude,@potholeId = b.PotholeID,@vehicleCrossed = b.VehiclesCrossed,@latestTimestamp = b.LatestTimestamp,@totalDepthRecorded = b.TotalDepthRecorded 

FROM   inserted a,PotholeSummary\_T b   
WHERE  a.RoadID = b.RoadID   
           *-- this is to check whether new pothole events fits in the existing summary*   
           *-- or its a new pothole which we have to insert*   
AND (1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MinLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MinLatitude)) \* power((sin(radians((b.MinLongitude - a.PLongitude) / 2))),2))) < 10.0  
       OR     1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MaxLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MaxLatitude)) \* power((sin(radians((b.MinLongitude - a.PLongitude) / 2))),2))) < 10.0  
       OR     1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MinLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MinLatitude)) \* power((sin(radians((b.MaxLongitude - a.PLongitude) / 2))),2))) < 10.0  
       OR     1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MaxLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MaxLatitude)) \* power((sin(radians((b.MaxLongitude - a.PLongitude) / 2))),2))) < 10.0  )   
  
    IF @pLat < @minLat   
      SET @minLat = @pLat   
  
    IF @pLat > @maxLat   
      SET @maxLat = @pLat   
  
    IF @pLong < @minLong   
      SET @minLong = @pLong   
  
    IF @pLong > @maxLong   
      SET @maxLong = @pLong   
  
    IF *Datediff*(second, @pTimestamp, @latestTimestamp) > 0   
      SET @pTimestamp = @latestTimestamp   
  
    IF @pLat IS NOT NULL   
      BEGIN   
          PRINT 'updated row  :'   
                + *Cast* (@potholeId AS *VARCHAR*(100))   
  
          UPDATE PotholeSummary\_T   
          SET    MinLatitude = @minLat,MaxLatitude = @maxLat, MinLongitude = @minLong, MaxLongitude = @maxLong,AvgPotholeDepth = ( @totalDepthRecorded + @pDepth) / (@vehicleCrossed + 1), VehiclesCrossed = @vehicleCrossed + 1, LatestTimestamp = @pTimestamp, TotalDepthRecorded = @totalDepthRecorded + @pDepth 

WHERE  PotholeID = @potholeId   
      END   
    ELSE   
      BEGIN   
          PRINT 'new row inserted'   
  
          SELECT @pEventId = inserted.PotholeEventID   
          FROM   inserted   
  
          INSERT INTO PotholeSummary\_T   
          SELECT inserted.PLatitude,inserted.PLatitude,inserted.PLongitude,   
                 inserted.PLongitude,   
                 inserted.PotholeDepth,1,*-- vehicles crossed,*   
                 inserted.PotholeDepth,inserted.Timestamp,inserted.RoadID   
          FROM   inserted   
  
          SELECT @potholeId = PotholeSummary\_T.PotholeID   
          FROM   PotholeSummary\_T,inserted   
          WHERE  PotholeSummary\_T.MinLatitude = inserted.PLatitude   
                 AND PotholeSummary\_T.MaxLatitude = inserted.PLatitude   
                 AND PotholeSummary\_T.MinLongitude = inserted.PLongitude   
                 AND PotholeSummary\_T.MaxLongitude = inserted.PLongitude   
                 AND PotholeSummary\_T.RoadID = inserted.RoadID   
      END   
  
    UPDATE PotholeEvent\_T   
    SET    PotholeID = @potholeId   
    WHERE  PotholeEventID = @pEventId

## Trigger - Location Event

The application will track the user’s location such as latitude, longitude and every event gets recorded in **location\_event** table. On every location event entry, trigger will execute to check if user is coming closer to pothole. If user comes with 1 km range of a pothole then trigger will fire an alert in the the alert\_table.

CREATE TRIGGER [dbo].[Trigger2]   
ON [dbo].[LocationEvent\_T]   
AFTER INSERT   
AS   
    DECLARE @lLat *decimal*(10, 8), @lLong *decimal*(11, 8), @lTimestamp  *datetime*, @roadId  *bigint*,  @userId          *bigint*, @StatusId  *bigint*,  @locationEventId *bigint*, @potholeId  *bigint*,  @minLat *decimal*(10, 8),  @maxLat  *decimal*(10, 8),  @minLong *decimal*(11, 8),  @maxLong  *decimal*(11, 8),  @latestTimestamp *datetime*   
  
    SELECT @lLat = a.LocLatitude,  @lLong = a.LocLongitude,  @lTimestamp = a.LocTimeStamp,  @userId = a.UserID,  @locationEventId = a.LocationEventID,  @minLat = b.MinLatitude, @maxLat = b.MaxLatitude,  @minLong = b.MinLongitude,  @maxLong = b.MaxLongitude,  @potholeId = b.PotholeID

FROM   inserted a,  PotholeSummary\_T b   
WHERE  a.RoadID = b.RoadID   
AND ( 1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MinLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MinLatitude)) \* power((sin(radians((b.MinLongitude - a.PLongitude) / 2))),2))) < 1000.0  
       OR     1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MaxLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MaxLatitude)) \* power((sin(radians((b.MinLongitude - a.PLongitude) / 2))),2))) < 1000.0  
       OR     1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MinLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MinLatitude)) \* power((sin(radians((b.MaxLongitude - a.PLongitude) / 2))),2))) < 1000.0  
       OR     1609.34 \* 2 \* 3961 \* asin(sqrt(power((sin(radians((b.MaxLatitude - a.PLatitude) / 2))), 2) + cos(radians(a.PLatitude)) \* cos(radians(b.MaxLatitude)) \* power((sin(radians((b.MaxLongitude - a.PLongitude) / 2))),2))) < 1000.0 )

IF @lLat IS NOT NULL   
 BEGIN   
  SELECT @StatusId = StatusID   
  FROM   RepairWork\_T   
  WHERE  @potholeId = RepairWork\_T.PotholeID

PRINT 'new row inserted'

IF @StatusId IS NULL OR @StatusId = 1   
   INSERT INTO PotholeAlert\_T VALUES('Pothole open, please drive carefully',  @userId,  @locationEventId,  @potholeId) 

ELSE IF @StatusId = 2 INSERT INTO PotholeAlert\_T VALUES     ('Pothole repair work is in progress, please drive carefully' , @userId,  @locationEventId,  @potholeId) 

END

## Trigger – Status Change Log

This trigger keep track of the status id which gets updated in the repair work, whenever the status id is updated, a row gets inserted in the **StatusUpdates\_Log** table. This can assist in knowing what is the current status of the repair work and when the pothole was closed.

CREATE TRIGGER [dbo].[StatusUpdate]   
ON [dbo].[RepairWork\_T]   
FOR UPDATE, INSERT   
AS   
    IF UPDATE(StatusID)   
      BEGIN   
          INSERT INTO StatusUpdates\_Log   
                      (StatusID,   
                       NewStatusID,   
                       RepairID,   
                       UpdateTimeStamp)   
          SELECT deleted.StatusID,   
                 inserted.StatusID,   
                 inserted.RepairID,   
                 *Getdate*()   
          FROM   inserted,   
                 deleted   
          WHERE  deleted.RepairID = inserted.RepairID   
      END

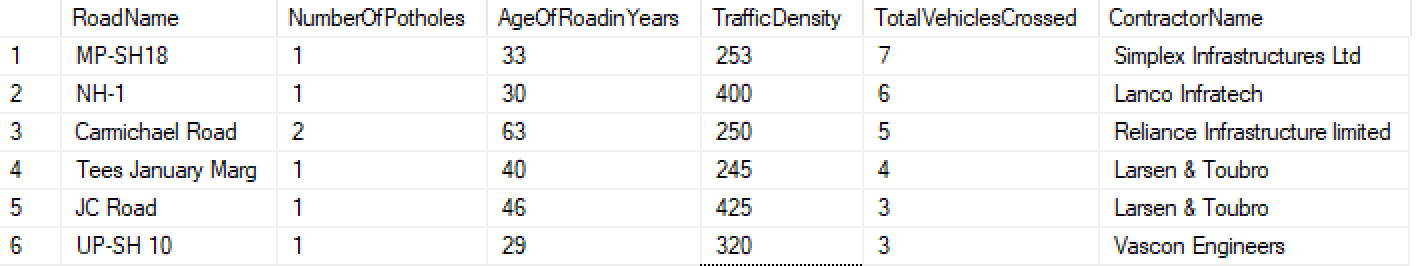
## Trigger – Password Length Check

This trigger keep track of the status id which gets updated in the repair work, whenever the status id is updated, a row gets inserted in the **StatusUpdates\_Log** table. This can assist in knowing what is the current status of the repair work and when the pothole was closed.

CREATE TRIGGER [dbo].[PasswordCheck]   
ON [dbo].[User\_T]   
FOR INSERT   
AS   
DECLARE @password *nvarchar*(max)   
SELECT @password = inserted.password FROM   inserted   
  
IF( *Len*(@password) <= 8 )   
BEGIN   
 PRINT 'Password should be more than 8 characters'   
 ROLLBACK   
END

# Views

## Pothole Repair Priority View - For Jurisdiction(Business User)



Jurisdiction can decide on the basis of this view, which pothole should be given more priority for the repair as here they can see age of road, traffic density and total Vehicle crossed and can also see the performance of the contractor like which contractor roads get potholes frequently

CREATE TABLE PotholeRepairPriority\_View   
  (   
     RoadName *nvarchar*(max) NOT NULL,NumberOfPotholes *int*,AgeOfRoadinYears *int*,   
     TrafficDensity *int*,TotalVehiclesCrossed *int*,ContractorName *nvarchar*(max)   
  )   
  
CREATE PROCEDURE **RefreshPotholeRepairPriority\_View**   
AS   
    DELETE FROM PotholeRepairPriority\_View   
  
    INSERT INTO PotholeRepairPriority\_View   
    SELECT r.RoadName,*Count*(DISTINCT p.PotholeID) AS NumberOfPotholes,   
           *Datediff*(year, r.DateOfConstructionCompleted, '11/23/2018')AS   
           AgeOfRoadinYears,r.TrafficDensity, *Sum*(p.VehiclesCrossed) AS TotalVehiclesCrossed,c.ContractorName   
    FROM   Road\_T r,PotholeSummary\_T p,Contractor\_T c   
    WHERE  r.RoadID = p.RoadID   
           AND r.ContractorID = c.ContractorID   
    GROUP  BY r.RoadName,r.DateOfConstructionCompleted,r.TrafficDensity,   
    c.ContractorName   
    ORDER  BY AgeOfRoadinYears DESC,TotalVehiclesCrossed DESC   
  
EXECUTE **RefreshPotholeRepairPriority\_View** 

SELECT \* FROM   PotholeRepairPriority\_View

## Quality Inspector Performance View - For Jurisdiction(Business User)

From this view, Jurisdiction can see the performance of their quality inspectors as they can see the number of days since the pothole has been registered and have been put for repair and the status of repair has still changed.

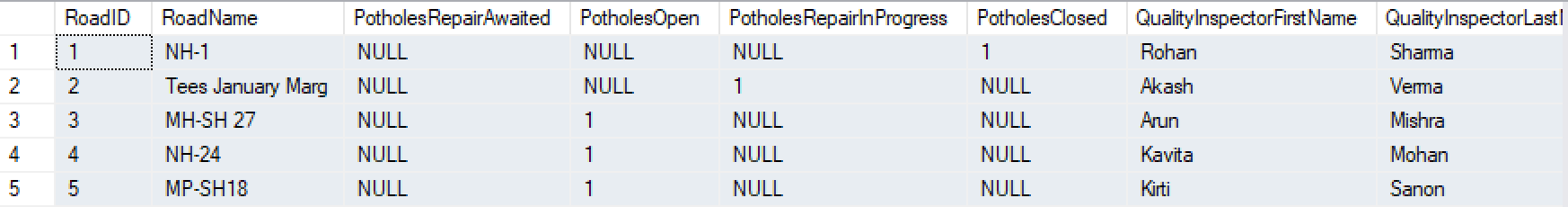


CREATE TABLE QualityInspectorPerformance\_View (RoadName *nvarchar*(max),PotholeID *bigint*,Open\_SinceinDays *int*,QualityInspectorFirstName *nvarchar*(max),QualityInspectorLastName *nvarchar*(max))   
  
CREATE PROCEDURE **RefreshQualityInspectorPerformance\_View**   
AS   
    DELETE FROM QualityInspectorPerformance\_View   
  
    INSERT INTO QualityInspectorPerformance\_View   
    SELECT r.RoadName,p.PotholeID,*Datediff*(day, p.LatestTimestamp,t.RepairStartDate) AS Open\_SinceinDays, q.QFirstName AS QualityInspectorFirstName, q.QLastname AS QualityInspectorLastName   
    FROM   Road\_T r,PotholeSummary\_T p,RepairWork\_T t,QualityInspector\_T q   
    WHERE  r.RoadID = p.PotholeID   
           AND t.PotholeID = p.PotholeID   
           AND q.QInspectorID = t.QInspectorID   
           AND t.StatusID = 1   
    ORDER  BY Open\_SinceinDays DESC;   
  
EXECUTE **RefreshQualityInspectorPerformance\_View**   
SELECT \* FROM QualityInspectorPerformance\_View

## Pothole Repair Status View - Quality Inspector(Business User)

This view is for the Quality Inspector where he can see how many pothole are open, how many are repair in progress and how many have been closed under his Supervision and

How many potholes are still awaiting action, that means they are registered but not put up for repair

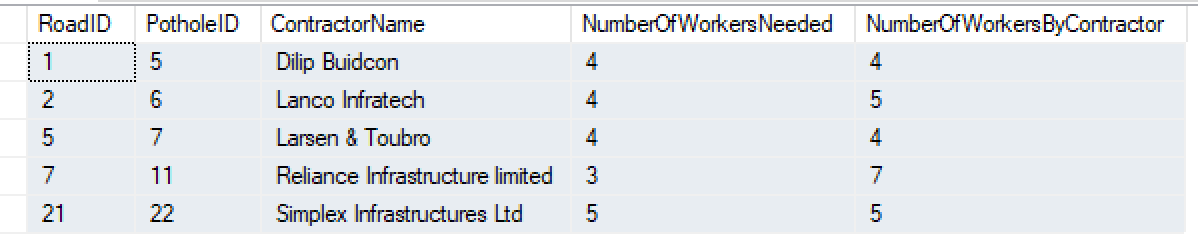


CREATE TABLE PotholeRepairStatus\_View   
  (   
     RoadID *bigint*,RoadName *nvarchar*(max),PotholesRepairAwaited *int*,   
     PotholesOpen *int*,PotholesRepairInProgress *int*,PotholesClosed *int*,  QualityInspectorFirstName *nvarchar*(max), QualityInspectorLastName *nvarchar*(max)   
  )   
  
CREATE PROCEDURE **RefreshPotholeRepairStatus\_View**   
AS   
    DELETE FROM PotholeRepairStatus\_View   
  
    INSERT INTO PotholeRepairStatus\_View   
    SELECT r.RoadID,r.RoadName,d.NumberOfPotholesRepairAwaited AS PotholesRepairAwaited, a.NumberOfPotholesOpen AS PotholesOpen, b.NumberOfPotholeRepairInProgress AS PotholeRepairInProgress, c.NumberOfPotholesClosed AS PotholesClosed, e.QFirstName AS QualityInspectorFirstName,e.QlastName AS QualityInspectorLastName  
    FROM   Road\_T r   
    LEFT JOIN (SELECT p.RoadID,*Count*(t.StatusID) AS NumberOfPotholesOpen FROM   PotholeSummary\_T p,Status\_T s,RepairWork\_T t WHERE  p.PotholeID = t.PotholeIDAND t.StatusID = s.StatusID AND t.StatusID = 1 GROUP  BY p.RoadID) a ON r.RoadID = a.RoadID   
    LEFT JOIN (SELECT r.RoadID,*Count*(t.StatusID) AS NumberOfPotholeRepairInProgress FROM   Road\_T r,PotholeSummary\_T p,RepairWork\_T t WHERE  t.PotholeID = p.PotholeID AND r.RoadID = p.RoadID AND t.StatusID = 2 GROUP  BY r.RoadID) b ON r.RoadID = b.RoadID   
  LEFT JOIN(SELECT r.RoadID,*Count*(t.StatusID) AS NumberOfPotholesClosed FROM   Road\_T r,PotholeSummary\_T p,RepairWork\_T t WHERE  t.PotholeID = p.PotholeID AND r.RoadID = p.RoadID AND t.StatusID = 3 GROUP  BY r.RoadID) AS c ON r.RoadID = c.RoadID   
    LEFT JOIN (SELECT r.RoadID,*Count*(p.PotholeID) AS NumberOfPotholesRepairAwaited FROM   PotholeSummary\_T p,Road\_T r WHERE  r.RoadID = p.RoadID AND p.PotholeID NOT IN (SELECT PotholeID FROM RepairWork\_T)

GROUP  BY r.RoadID) d   
   ON r.RoadID = d.RoadID   
    LEFT JOIN (SELECT r.RoadID,QFirstName,QLastname FROM   QualityInspector\_T q,Road\_T r,RepairWork\_T t,PotholeSummary\_T p WHERE  q.QInspectorID = t.QInspectorID AND r.RoadID = p.RoadID AND t.PotholeID = p.PotholeID) e   
   ON r.RoadID = e.RoadID   
  
EXECUTE **RefreshPotholeRepairStatus\_View**   
SELECT \* FROM PotholeRepairStatus\_View

# Quality Inspector Supervision View - Quality Inspector(Business User)

From this view, The Quality inspector can recognize how seriously the contractor is taking the repair works, as he/she can see Number of workers needed for a repair project and how much the contractor is providing. As less workers means more time to repair the pothole.



CREATE TABLE QualityInspectorSupervision\_View (   
     RoadID *bigint*,PotholeID *bigint*,ContractorName *nvarchar*(max),   
     NumberOfWorkersNeeded *int*,NumberOfWorkersByContractor *int*   
  )   
  
CREATE PROCEDURE **RefreshQualityInspectorSupervision\_View**   
AS   
    DELETE FROM QualityInspectorSupervision\_View   
 INSERT INTO QualityInspectorSupervision\_View   
    SELECT p.RoadID,p.PotholeID,c.ContractorName,r.NumberOfWorkers AS NumberOfWorkersNeeded ,*Count*(DISTINCT w.WorkerID) AS NumberOfWorkersByContractor   
    FROM   RepairWork\_T r,RepairWorkers\_T w,PotholeSummary\_T p,Contractor\_T c   
    WHERE  p.PotholeID = r.PotholeID   
    AND w.ContractorID = c.ContractorID   
    AND w.RepairID = r.RepairID   
    GROUP  BY p.RoadID,p.PotholeID,c.ContractorName,r.NumberOfWorkers   
  
EXECUTE **RefreshQualityInspectorSupervision\_View**   
SELECT \* FROM QualityInspectorSupervision\_View