September 2017

PHASE 1

PENNDOT EXTREME WEATHER VULNERABILITY STUDY

ADDENDUM 1

UPDATES TO HISTORICAL RISK ASSESSMENT

pennsylvania
DEPARTMENT OF TRANSPORTATION

The Pennsylvania Department of Transportation (PennDOT) has initiated a multi-phase effort aimed to better anticipate the consequences and impacts of extreme weather events and to identify funding priorities and strategies to improve transportation system resiliency.

In April 2017, PennDOT released a *Phase 1 Extreme Weather Vulnerability Study* that focused on the evaluation of historic vulnerabilities, development of a framework for addressing climate change impacts, and an initial assessment of risks and priorities related to the identified vulnerabilities. The study's analyses and mapping products are focused primarily on the flooding impacts on state-owned roads and bridges.

Addendum 1 includes updates to the original study calculations and mapping as completed in September of 2017. It includes the use of more recent flooding closure data from PennDOT's Road Condition Reporting System (RCRS), revised risk assessment criteria, updated mapping and data assessments, and the development of automated program scripts to execute risk assessment calculations and results.

Addendum 1 Updates

- Newer RCRS Flooding Closure Data
- Revised Prioritization Criteria
- Updating Mapping
- Automated Process Scripting

The revisions for Addendum 1 are focused on the historical flooding assessment. No changes have been made to the forecast flooding assessment and risk analysis as included in the original report.

RCRS Closure Data

The historic flooding assessment in Addendum 1 utilizes an updated download of PennDOT's Road Conditions Reporting System (RCRS) with flooding closure data records from November 16, 2006 through August 19, 2017.

Location information on flooding closures is provided in different forms within the database. The original study processed records solely on specified PennDOT state route and segment number fields within the RCRS database. This addendum expands on the location identification process using additional methodologies and data fields. The new process expands the number of flooding locations and events included in the risk prioritization and mapping products.

Risk Assessment

The *Phase 1 Extreme Weather Vulnerability Study* includes a risk assessment framework to evaluate and prioritize historic and future flooding vulnerabilities within Pennsylvania. The risk assessment integrated information from PennDOT's Roadway Management System (RMS), Bridge Management System, and RMS Pipe databases. The criteria were scored and weighted, providing an overall score for each flooding location.

Addendum 1 has revised the historic risk assessment results and mapping. The forecast impacts of climate change on risks and the associated mapping remains unchanged from the original report. The historical risk revisions include analysis of the expanded number of RCRS flooding locations, the use of a new data source for historic rainfall data, and the addition of a flooding frequency variable to the risk scoring calculation.

The RCRS contains a number of records related to any flooding closure. In many cases, multiple records may be included for the same event representing status updates on the closed roadway. A process has been implemented to identify unique flooding events which provides an event frequency number for each location. This new data field has been integrated into the historic risk assessment criteria as illustrated in the following tables.

Historic Risk Scoring Formula

Category	Weight	Criteria & Weights for Scoring
Exposure	30%	Precipitation score x 0.3
		Floodplain score x 0.3
		Flooding Frequency x 0.4
Sensitivity	30%	OPI Score x 0.4
		Scour Score x 0.4
		Pipe Score x 0.2
Consequence	40%	Volume Score x 0.4
		Functional Class Score x 0.3
		Detour Score x 0.1
		Truck Volume Score x 0.2

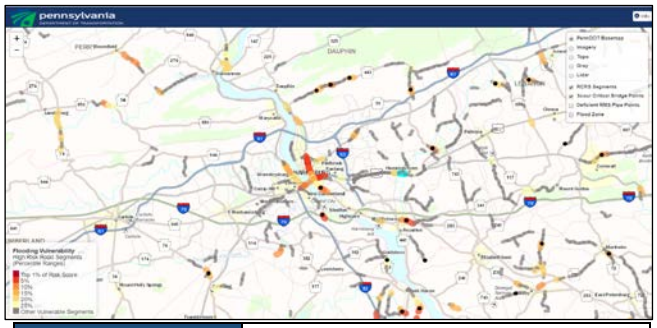
Risk Criteria Scoring Definitions

Variable	Scoring Values
Precipitation (4-day Rainfall)	• ≤ 1.5 inches = 10 • ≤3 = 7.5 • ≤5 = 5 • ≤7 = 2.5 • >7 = 1
Floodplain	• Yes = 10 ; No = 0
Frequency of Flooding in RCRS History	• > 12 times historically = 10 • > 7 = 8 • > 5 = 6 • > 1 = 4 • ≤ 1 = 2
Overall Pavement Index (OPI)	• Excellent = 2.5 • Good, Other = 5.0 • Fair = 7.5 • Poor = 10.0
Scour Critical Bridges	• Yes = 10 ; No = 0
Deficient Pipes	• Yes = 10 ; No = 0
Two-Way Traffic Volume	• < 5,000 = 1 • < 10,000 = 2 • < 15,000 = 3 • < 20,000 = 4 • < 25,000 = 5
Two-Way Truck Volume	• < 500 = 1 • < 1,000 = 2 • < 2,000 = 3 • < 3,000 = 4 • < 4,000 = 5
Functional Class	• Local = 1 • Collector/Minor Arterial = 2.5 • Other = 5 • Principal Arterial = 7.5 • Interstate = 10
Included as PennDOT Detour Route	• Yes = 10 ; No = 0

Addendum 1 Mapping Products

The historic risk assessment mapping has been updated and integrated into several online map tools. These tools allow users to visualize the risk assessment results and to access information on each of the flooding locations. The maps illustrate the risk scores as percentile groups (e.g. the top 1%, 5%, 10%, etc. of flooding risk locations in state). GIS overlays are provided for individual scour critical bridges and deficient pipes that are located within the vulnerable roadway segments. Selecting the roadway segments allows users to see information on variable scoring and to obtain other information on RCRS record details as well as rainfall data within the watershed.

Addendum 1 Risk Assessment Mapping Products



Risk Mapping (All Locations)	Map Link
Risk Mapping (Locations with > 1 Closure Events)	Map Link (Map provided as an alternative to above with less flooding locations)

Maps require the use of the Google Chrome browser. Some features may not work properly using Internet Explorer.

As with the original study, PennDOT will be integrating the above mapping details into their Pennshare mapping services. These maps are provided as additional resources. Using the "INFO" tab in the upper right hand corner of each map allows for the download of GIS "Shapefiles".

Data Sources Summary

The analyses and mapping for Addendum 1 has utilized information from PennDOT, the National Oceanic and Atmospheric Administration (NOAA), and other available sources as summarized in the following table.

Addendum 1 Data Sources

Name	Description
RCRS	Road Closure Reporting System
Source:	Obtained from PennDOT (Format specific)
Link to File Used:	Link to File
BMS	Bridge Database
Source:	Link to Source
Link to File Used:	Link to File
RMS ADMIN	Roadway Management System Data
Source:	Link to Source
Link to File Used:	Link to File
RMS SEG	Roadway Management System Data
Source:	Link to Source
Link to File Used:	Link to File
RMS TRAFFIC	Roadway Management System Data
Source:	Link to Source
Link to File Used:	Link to File
RMS PIPE	Pipe and Culvert Database
Source:	Link to Source
Link to File Used:	Link to File
Detour Routes	PennDOT Designated Detour Routes
Source:	Obtained from PennDOT (Format specific)
Link to File Used:	Link to File
Floodplains	Statewide Floodplain Polygons (PSU)
Source:	Link to Source
Link to File Used:	Link to File
Rainfall	NOAA Historic Rainfall (2005-2017)
Source:	Link to Source
Link to File Used:	Link to File
Gauge Points	NOAA Monitoring Station Locations
Source:	Link to Source
Link to File Used:	Link to File
Watershed	USGS HUC 8 Watershed Polygons
Source:	Link to Source
Link to File Used:	Link to File

Process Automation Scripting

One of key implementation steps identified in the original study is the integration of risk calculations within PennDOT's data systems. Risk assessments are based on specific roadway, bridge and pipe data which are constantly changing. In addition, new closure records are continually added to the RCRS system based on flooding events occurring within the state. Creating a process that can automatically be updated at periodic intervals will be valuable for PennDOT in monitoring and planning for future flooding events and assessing potential locations for resiliency strategies.

As an initial step, Addendum 1 has included the development of custom processing scripts to automate the calculations and mapping of flooding risks within the state based on available PennDOT databases and NOAA rainfall. These custom scripts were developed using the PostgreSQL data management system and the Python programming language. PostgreSQL is a general purpose and object-relational database management system that is free and open source software. Its source code is available under PostgreSQL license, a liberal open source license that allows for users to use, modify and distribute PostgreSQL in any form. Python is an interpreted, object-oriented, high-level programming language. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Appendix 1 provides the complete scripting code documenting the procedures and methods used to extract RCRS flooding records and calculate risk assessment scores using the data sources summarized above. The file and script is also available at the following link:

http://s3.amazonaws.com/tmp-map/climate/model/data-model.html

Comments on the Study

PennDOT's assessments of flooding vulnerability and risks should be considered to be an evolving process. Comments and insights from PennDOT district offices and other planning partners will be used to refine and improve the vulnerability locations and risk assessment methods. Any comments can be directed to:

Douglas E. Zimmerman
Assistant for Strategic Management
PennDOT Office of Planning
400 North Street | 8th Floor
Harrisburg PA 17120
Phone: 717-787-0782
dozimmerma@pa.gov I www.penndot.gov

APPENDIX 1

PostgreSQL and Python Scripting for Flooding Risk Calculations

Process Script for Pennsylvania Extreme Weather Vulnerability Study

Addendum 1 - September 2017

- Import raw RCRS data from .xls provided by PennDOT into a table called **rcrs_src_data**.
- The query listed in the source spreadsheet from PennDOT is:
  select * from closures_all_records_v where causeid = '7' order by cty_code_bgn, st_rt_no

```
DROP TABLE IF EXISTS rcrs_src_data;

CREATE TABLE rcrs_src_data(
eventid character varying(50),
st_rt_no character varying(50),
juris character varying(50),
cty_code_bgn character varying(50),
muni_bgn character varying(50),
street_bgn character varying(50),
mile_bgn character varying(50),
exit_bgn character varying(50),
seg_bgn character varying(50),
offset_bgn character varying(50),
intersec_bgn character varying(50),
cty_code_end character varying(50),
muni_end character varying(50),
street_end character varying(50),
mile_end character varying(50),
exit_end character varying(50),
seg_end character varying(50),
offset_end character varying(50),
inter_end character varying(50),
cause_id character varying(50),
flood character varying(50),
wind character varying(50),
erosion character varying(50),
lndslide character varying(50),
ice_snow character varying(50),
lighting character varying(50),
misc character varying(50),
desc_incident character varying(1750),
date_closed character varying(50),
est_time_open character varying(50),
last_edit character varying(50),
foo character varying(50)
);
/*
Import raw csv data from RCRS export.
*/
```

```
COPY rcrs_src_data FROM 'C:\prj\pa-climate\data-model\src-shp\rcrs-src-data\rcrs-june8-2016.csv' DELIMITER ',' CSV HEADER;
```

- Create uniform LRS format for RMSSEG canidate extraction.
- This SQL searches the RCRS raw data for records with seg_bgn and seg_end or splits street_bgn & street_end fields into County, Route, Segment and Offset fields.

```
-- create uniform LRS format for RMSSEG canidate extraction

DROP TABLE IF EXISTS rcrs_src_clean;

-- RCRS data w/ standards LRS fields (cty, sr, seg)

CREATE TABLE rcrs_src_clean AS
SELECT
DISTINCT ON (eventid)
eventid,
cty_code_bgn AS cty_code,
st_rt_no,
lpad(seg_bgn, 4, '0') AS seg_bgn,
lpad(seg_end, 4, '0') AS seg_end,
date_time_closed::DATE
FROM rcrs_src_data
WHERE
street_bgn IS NULL
AND
street_end IS NULL
AND
cty_code_bgn IS NOT NULL
AND
cty_code_end IS NOT NULL
AND
seg_bgn IS NOT NULL
AND
seg_end IS NOT NULL

UNION

-- ADD just street begin segments
-- Records w/ only street_bgn and NOT street_end
SELECT DISTINCT ON (eventid)
eventid,
substring(street_bgn,1,2) AS cty_code,
substring(street_bgn,4,4) AS st_rt_no,
substring(street_bgn,9,4) AS seg_bgn,
substring(street_bgn,9,4) AS seg_end,
--street_bgn, street_end,
date_time_closed::DATE
FROM rcrs_src_data
WHERE
```

```
street_bgn IS NOT NULL
and
street_end IS NULL

UNION

-- ADD just street end segments
-- NO street_bgn

SELECT DISTINCT ON (eventid)
eventid,
substring(street_end,1,2) AS cty_code,
substring(street_end,4,4) AS st_rt_no,
substring(street_end,9,4) AS seg_bgn,
substring(street_end,9,4) AS seg_end,
--street_bgn, street_end,
date_time_closed::DATE
FROM rcrs_src_data
WHERE
street_end IS NOT NULL
and
street_bgn IS NULL

UNION

-- Street end & Street begin fields available

SELECT DISTINCT ON (eventid)
eventid,
substring(street_bgn,1,2) AS cty_code,
substring(street_bgn,4,4) AS st_rt_no,
substring(street_bgn,9,4) AS seg_bgn,
substring(street_end,9,4) AS seg_end,
--street_bgn, street_end,
date_time_closed::DATE
FROM rcrs_src_data
WHERE
street_bgn IS NOT NULL
AND
street_end IS NOT NULL


UNION

-- ADD Intersection bgn / end records

SELECT eventid,
substring(intersr_bgn,1,2) AS cty_code,
substring(intersr_bgn,4,4) AS st_rt_no,
substring(intersr_bgn,9,4) AS seg_bgn,
substring(intersr_end,9,4) AS seg_end,
date_time_closed::DATE
FROM public.rcrs_src_data
WHERE
```

```
street_bgn IS NULL
AND
street_end IS NULL
AND
seg_bgn IS NULL
AND
intersr_bgn IS NOT NULL
AND
intersr_end IS NOT NULL



UNION

-- ADD Intersection bgn only records

SELECT eventid,
substring(intersr_bgn,1,2) AS cty_code,
substring(intersr_bgn,4,4) AS st_rt_no,
substring(intersr_bgn,9,4) AS seg_bgn,
substring(intersr_bgn,9,4) AS seg_end,
date_time_closed::DATE
FROM public.rcrs_src_data
WHERE
street_bgn IS NULL
AND
street_end IS NULL
AND
seg_bgn IS NULL
AND
intersr_bgn IS NOT NULL
AND
intersr_end IS  NULL



UNION


-- ADD Intersection end only records

SELECT eventid,
substring(intersr_end,1,2) AS cty_code,
substring(intersr_end,4,4) AS st_rt_no,
substring(intersr_end,9,4) AS seg_bgn,
substring(intersr_end,9,4) AS seg_end,
date_time_closed::DATE
FROM public.rcrs_src_data
WHERE
street_bgn IS NULL
AND
street_end IS NULL
AND
seg_bgn IS NULL
```

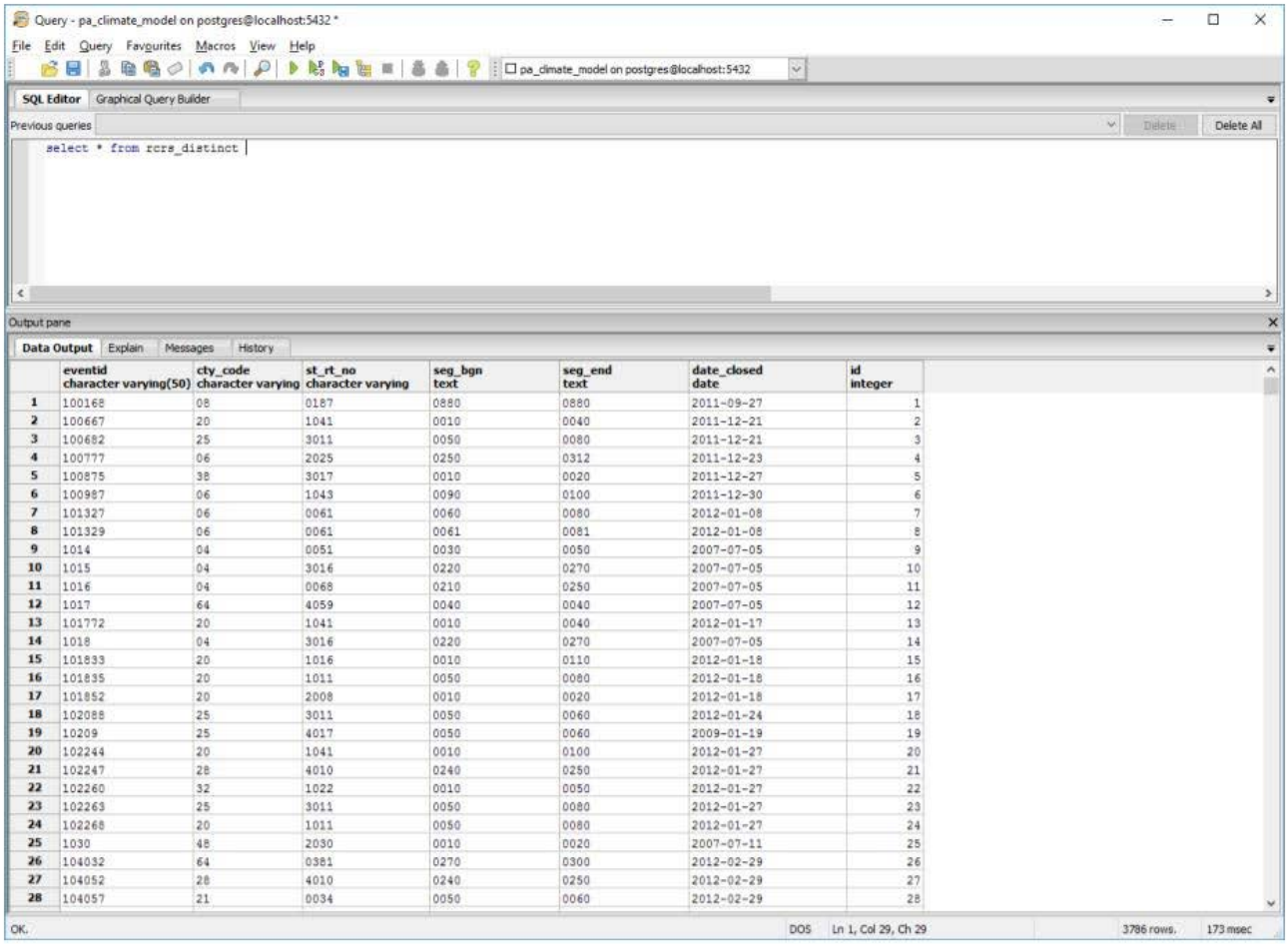
```
AND
intersr_bgn IS  NULL
AND
intersr_end IS  NOT NULL


UNION

-- ADD EXIT LRS records
-- Both exit_bgn and exit_end have valid keys

SELECT eventid,
substring(exit_bgn,1,2) AS cty_code,
substring(exit_bgn,4,4) AS st_rt_no,
substring(exit_bgn,11,4) AS seg_bgn,
substring(exit_end,11,4) AS seg_end,
date_time_closed::DATE
FROM public.rcrs_src_data
WHERE
exit_bgn IS NOT NULL
AND
exit_bgn <> '0_1'
AND
exit_end IS NOT NULL
AND
exit_end <> '0_1'


UNION

-- ONLY exit_bgn keys

SELECT eventid,
substring(exit_bgn,1,2) AS cty_code,
substring(exit_bgn,4,4) AS st_rt_no,
substring(exit_bgn,11,4) AS seg_bgn,
substring(exit_bgn,11,4) AS seg_end,
date_time_closed::DATE
FROM public.rcrs_src_data
WHERE
exit_bgn IS NOT NULL
AND
exit_bgn <> '0_1'
AND
exit_end IS NULL

UNION

-- ONLY exit_end keys

SELECT eventid,
substring(exit_end,1,2) AS cty_code,
substring(exit_end,4,4) AS st_rt_no,
substring(exit_end,11,4) AS seg_bgn,
```

```
substring(exit_end,11,4) AS seg_end,
date_time_closed::DATE
FROM public.rcrs_src_data
WHERE
exit_end IS NOT NULL
AND
exit_end <> '0_1'
AND
exit_bgn IS NULL



;

-- about 50 duplicate eventid in both LRS formats


DROP TABLE IF EXISTS rcrs_distinct;
CREATE TABLE rcrs_distinct
AS
SELECT
DISTINCT ON (eventid)
* FROM rcrs_src_clean;


-- add unique id
ALTER TABLE rcrs_distinct ADD COLUMN id SERIAL PRIMARY KEY;
```

- There are about 50 RCRS duplicate eventid's that have both a seg_bgn & seg_end and a street_bgn & street_end LRS format. The following SQL creates a table named **rcrs_distinct** and removes duplicate eventid's.

```
DROP TABLE IF EXISTS rcrs_distinct;
CREATE TABLE rcrs_distinct
AS
SELECT
DISTINCT ON (eventid)
* FROM rcrs_src_clean;

-- add unique id
ALTER TABLE rcrs_distinct ADD COLUMN id SERIAL PRIMARY KEY;
```

- The image below shows the data structure of the table **rcrs_distinct** with a unique eventid, cty_code, seg_bgn, seg_end and the date of the RCRS closure. At this point the goal is to get every **eventid & date** in the RCRS data that has valid LRS information (cty_code, st_rt_no, seg_bgn & seg_end).



	eventid character varying(50)	cty_code character varying	st_rt_no character varying	seg_bgn text	seg_end text	date_closed date	id integer
1	100168	08	0187	0880	0880	2011-09-27	1
2	100667	20	1041	0010	0040	2011-12-21	2
3	100682	25	3011	0050	0080	2011-12-21	3
4	100777	06	2025	0250	0312	2011-12-23	4
5	100875	38	3017	0010	0020	2011-12-27	5
6	100987	06	1043	0090	0100	2011-12-30	6
7	101327	06	0061	0060	0080	2012-01-08	7
8	101329	06	0061	0061	0081	2012-01-08	8
9	1014	04	0051	0030	0050	2007-07-05	9
10	1015	04	3016	0220	0270	2007-07-05	10
11	1016	04	0068	0210	0250	2007-07-05	11
12	1017	64	4059	0040	0040	2007-07-05	12
13	101772	20	1041	0010	0040	2012-01-17	13
14	1018	04	3016	0220	0270	2007-07-05	14
15	101833	20	1016	0010	0110	2012-01-18	15
16	101835	20	1011	0050	0080	2012-01-18	16
17	101852	20	2008	0010	0020	2012-01-18	17
18	102088	25	3011	0050	0060	2012-01-24	18
19	10209	25	4017	0050	0060	2009-01-19	19
20	102244	20	1041	0010	0100	2012-01-27	20
21	102247	28	4010	0240	0250	2012-01-27	21
22	102260	32	1022	0010	0050	2012-01-27	22
23	102263	25	3011	0050	0080	2012-01-27	23
24	102268	20	1011	0050	0080	2012-01-27	24
25	1030	48	2030	0010	0020	2007-07-11	25
26	104032	64	0381	0270	0300	2012-02-29	26
27	104052	28	4010	0240	0250	2012-02-29	27
28	104057	21	0034	0050	0060	2012-02-29	28

- The following SQL creates a table **rainfall_2005_2016** and loads the .csv of daily rainfall totals for each monitoring station.

```
DROP TABLE IF EXISTS rainfall_2005_2016;

CREATE TABLE rainfall_2005_2016
(
station character varying(50),
dtd date,
prcp character varying(50),
huc_8 character varying(50)

);
```

```
COPY rainfall_2005_2016 FROM 'C:\prj\pa-climate\data-model\src-shp\rainfall-2005-
2016\rainfall-2005-2016.csv' DELIMITER ',' CSV HEADER;

-- add unique id
ALTER TABLE rainfall_2005_2016 ADD COLUMN id SERIAL PRIMARY KEY;

CREATE INDEX ON rainfall_2005_2016 (station);
CREATE INDEX ON rainfall_2005_2016 (dtd);
```

- The following SQL creates a table **rcrs_rms_candidate_segment** and will contain all of the individual RMS segments that are in the RCRS database. So if a closure event occurs between segments 0010 and 0040, segments 0010-0020-0030 & 0040 are included.

```
DROP TABLE IF EXISTS rcrs_rms_candidate_segment;
CREATE TABLE rcrs_rms_candidate_segment
(
eventid character varying,
date_closed character varying,
rms_key character varying,
the_geom geometry(MultiLineString,4326)
);

ALTER TABLE rcrs_rms_candidate_segment ADD COLUMN id SERIAL PRIMARY KEY;

 ALTER TABLE rms_seg ADD COLUMN rms_key varchar(12);
 UPDATE rms_seg SET rms_key = cty_code || st_rt_no || seg_no;
 CREATE INDEX ON rms_seg (rms_key);
 CREATE INDEX ON rms_seg (cty_code);
 CREATE INDEX ON rms_seg (st_rt_no);
 CREATE INDEX ON rms_seg (seg_no);
```

- The following Python script cycles through each record in the **rcrs_distinct** table, finds each of the segments that exists between the seg_bgn and seg_end fields and inserts the results into the **rcrs_rms_candidate_segment** table.

```
import psycopg2
import psycopg2.extras



def get_rmsseg_values(eventid, cty_code, st_rt_no, seg_bgn, seg_end, date_closed):

        sql2 = '''
        insert into rcrs_rms_candidate_segment
        select
        '%s' as eventid,
```

```
        '%s' as date_closed,
        --cty_code || st_rt_no || seg_no as rms_key,
        rms_key,
        the_geom
        from
        rms_seg
        where cty_code = '%s'
        and
        st_rt_no = '%s'
        and
        seg_no >= '%s'
        and
        seg_no <= '%s';
        '''%(eventid, date_closed, cty_code, st_rt_no,seg_bgn,seg_end)

        #print sql2
        print eventid


        cursor2 = conn.cursor()
        cursor2.execute(sql2)




def get_sql_list_of_rcrs():
        sql = '''
        select
        eventid, cty_code,
        st_rt_no, seg_bgn,
        seg_end, date_closed
        from
        rcrs_distinct
        '''
        conn = psycopg2.connect(conn_string)
        cursor = conn.cursor()
        cursor.execute(sql)
        results = cursor.fetchall()

        for a in results:
                eventid = a[0]
                cty_code = a[1]
                st_rt_no = a[2]
                seg_bgn = a[3]
                seg_end = a[4]
                date_closed = a[5]

                get_rmsseg_values(eventid, cty_code, st_rt_no, seg_bgn, seg_end,
date_closed)
```

```
if __name__ == '__main__':

        conn_string = "host='localhost' dbname='pa_climate_model' port='5432'
user='postgres' password='postgres'"
        conn = psycopg2.connect(conn_string)
        get_sql_list_of_rcrs()
        conn.commit()
```

- Add HUC 8 watershed code to the table **rcrs_rms_candidate_segment**

```
ALTER TABLE rcrs_rms_candidate_segment DROP COLUMN IF EXISTS huc_8;
ALTER TABLE rcrs_rms_candidate_segment ADD COLUMN huc_8 varchar(10);

UPDATE rcrs_rms_candidate_segment b
SET
huc_8 = subquery.huc8_id
FROM
(
SELECT a.huc8_id, b.id
FROM
huc8_poly a,
rcrs_rms_candidate_segment  b
WHERE
st_intersects(ST_LineInterpolatePoint(ST_GeometryN(b.the_geom,1),.5),a.the_geom)
) as subquery
WHERE
b.id = subquery.id;
```

- Add rainfall count fields to the table **rcrs_rms_candidate_segment.** These are place holder fields for the python script that will update the total rainfall within a HUC 8 watershed based on the date of the closure.
- This also adds **rain_count_m1**,**rain_count_m2 & rain_count_m3** for the date of the closure and the previous three days of rainfall values.

```
ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_count,
ADD COLUMN rain_count varchar(10);

ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_count_m1,
ADD COLUMN rain_count_m1 varchar(10);

ALTER TABLE rcrs_rms_candidate_segment
```

```
DROP COLUMN IF EXISTS rain_count_m2,
ADD COLUMN rain_count_m2 varchar(10);

ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_count_m3,
ADD COLUMN rain_count_m3 varchar(10);



ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_avg,
ADD COLUMN rain_avg varchar(10);

ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_avg_m1,
ADD COLUMN rain_avg_m1 varchar(10);

ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_avg_m2,
ADD COLUMN rain_avg_m2 varchar(10);

ALTER TABLE rcrs_rms_candidate_segment
DROP COLUMN IF EXISTS rain_avg_m3,
ADD COLUMN rain_avg_m3 varchar(10);
```

- The following python code cycles through each record in the **rcrs_rms_candidate_segment** table, grabs the date of the closure and then calculates the average rainfall within that watershed for the closure date and the following three days.

```
import psycopg2
import psycopg2.extras
from datetime import datetime, timedelta


def get_rain_values_m3(id,date_closed,huc_8):

        sql2 = '''
        select
        count(*),
        CAST(avg(prcp::float) AS DECIMAL(10,2)) as the_avg
        from
        rainfall_2005_2016
        where
        dtd = '%s'::date - interval '3' day
        and huc_8 = '%s'
        and prcp NOT LIKE '-9999.000000'
        '''%(date_closed, huc_8)

        #print sql2


        cursor2 = conn.cursor()
        cursor2.execute(sql2)
        a = cursor2.fetchall()


        rain_obs_cnt = a[0][0]
        rain_avg = a[0][1]
        sql3 = '''
        update rcrs_rms_candidate_segment
        set rain_avg_m3 = '%s', rain_count_m3 = '%s'
        where
        id = '%s'
        '''%(rain_avg, rain_obs_cnt, id)


        print sql3

        cursor3 = conn.cursor()
        cursor3.execute(sql3)



def get_rain_values_m2(id,date_closed,huc_8):

        sql2 = '''
```

```
        select
        count(*),
        CAST(avg(prcp::float) AS DECIMAL(10,2)) as the_avg
        from
        rainfall_2005_2016
        where
        dtd = '%s'::date - interval '2' day
        and huc_8 = '%s'
        and prcp NOT LIKE '-9999.000000'
        '''%(date_closed, huc_8)

        #print sql2


        cursor2 = conn.cursor()
        cursor2.execute(sql2)
        a = cursor2.fetchall()


        rain_obs_cnt = a[0][0]
        rain_avg = a[0][1]
        sql3 = '''
        update rcrs_rms_candidate_segment
        set rain_avg_m2 = '%s', rain_count_m2 = '%s'
        where
        id = '%s'
        '''%(rain_avg, rain_obs_cnt, id)


        print sql3

        cursor3 = conn.cursor()
        cursor3.execute(sql3)


def get_rain_values_m1(id,date_closed,huc_8):

        sql2 = '''
        select
        count(*),
        CAST(avg(prcp::float) AS DECIMAL(10,2)) as the_avg
        from
        rainfall_2005_2016
        where
        dtd = '%s'::date - interval '1' day
        and huc_8 = '%s'
        and prcp NOT LIKE '-9999.000000'
        '''%(date_closed, huc_8)

        #print sql2
```

```
        cursor2 = conn.cursor()
        cursor2.execute(sql2)
        a = cursor2.fetchall()


        rain_obs_cnt = a[0][0]
        rain_avg = a[0][1]
        sql3 = '''
        update rcrs_rms_candidate_segment
        set rain_avg_m1 = '%s', rain_count_m1 = '%s'
        where
        id = '%s'
        '''%(rain_avg, rain_obs_cnt, id)


        print sql3

        cursor3 = conn.cursor()
        cursor3.execute(sql3)



def get_rain_values(id,date_closed,huc_8):

        sql2 = '''
        select
        count(*),
        CAST(avg(prcp::float) AS DECIMAL(10,2)) as the_avg
        from
        rainfall_2005_2016
        where
        dtd = '%s'
        and huc_8 = '%s'
        and prcp NOT LIKE '-9999.000000'
        '''%(date_closed, huc_8)

        #print sql2


        cursor2 = conn.cursor()
        cursor2.execute(sql2)
        a = cursor2.fetchall()


        rain_obs_cnt = a[0][0]
        rain_avg = a[0][1]
        sql3 = '''
        update rcrs_rms_candidate_segment
        set rain_avg = '%s', rain_count = '%s'
        where
        id = '%s'
        '''%(rain_avg, rain_obs_cnt, id)
```

```
        print sql3

        cursor3 = conn.cursor()
        cursor3.execute(sql3)




def get_sql_list_of_rcrs():
        sql = '''
        select id, date_closed, huc_8
        from
        rcrs_rms_candidate_segment

        '''

        cursor = conn.cursor()
        cursor.execute(sql)
        results = cursor.fetchall()

        for a in results:
                id = a[0]
                date_closed = a[1]
                huc_8 = a[2]


                get_rain_values(id,date_closed,huc_8)
                get_rain_values_m1(id,date_closed,huc_8)
                get_rain_values_m2(id,date_closed,huc_8)
                get_rain_values_m3(id,date_closed,huc_8)




if __name__ == '__main__':

        conn_string = "host='localhost' dbname='pa_climate_model' port='5432'
user='postgres' password='postgres'"
        conn = psycopg2.connect(conn_string)
        get_sql_list_of_rcrs()
        conn.commit()
```

- The following SQL updates the rain_avg fields to remove the 'None' values from the NOAA data and replaces them with 0.0, this is necessary when calculating values later.

```
--REMOVE 'None' from rain_avg calc  (125 records)
UPDATE rcrs_rms_candidate_segment
SET rain_avg = '0.00'
WHERE
rain_avg = 'None';

UPDATE rcrs_rms_candidate_segment
SET rain_avg_m1 = '0.00'
WHERE
rain_avg_m1 = 'None';

UPDATE rcrs_rms_candidate_segment
SET rain_avg_m2 = '0.00'
WHERE
rain_avg_m2 = 'None';

UPDATE rcrs_rms_candidate_segment
SET rain_avg_m3 = '0.00'
WHERE
rain_avg_m3 = 'None';
```

- The following SQL creates a new table **rcrs_rms_candidate_segment_flat** which groups the records by segment and calculates the average 1 day and 4 day rainfall totals for each segment. This process 'flattens' out multiple records for one segment combining the average rainfall scores across multiple days into just one geometry.

```
-- NEED to flatten record with average rainfal on segment


-- MAKE Flat table of segments (8430 unique segments)
DROP TABLE IF EXISTS rcrs_rms_candidate_segment_flat;
CREATE TABLE rcrs_rms_candidate_segment_flat AS
SELECT
rms_key, count(*) AS seg_count,
avg(rain_avg::float) AS avg_one_day,
--sum(COALESCE(rain_avg::float) + COALESCE(rain_avg_p1::float) +
COALESCE(rain_avg_p2::float) + COALESCE(rain_avg_p3::float)) /4 AS coales_four_day,
min(rain_avg::float) + min(rain_avg_p1::float) + min(rain_avg_p2::float) +
min(rain_avg_p3::float) /4 as avg_four_day
FROM
rcrs_rms_candidate_segment
GROUP BY rms_key;
```

```
 --select * from rcrs_rms_candidate_segment_flat;


-- ADD back the_geom
DROP TABLE IF EXISTS rcrs_rms_candidate_segment_flat_geom;
CREATE TABLE rcrs_rms_candidate_segment_flat_geom as
SELECT
DISTINCT ON (rms_key)
a.*, ST_GeometryN(b.the_geom, 1) AS the_geom
FROM
rcrs_rms_candidate_segment_flat a,
rcrs_rms_candidate_segment b
WHERE
a.rms_key = b.rms_key;


CREATE INDEX sidx_rcrs_rms_candidate_segment_flat_geom
  ON public.rcrs_rms_candidate_segment_flat_geom
  USING gist
  (the_geom);
```

- The following SQL snippets add fields for the model components and their corresponding score values.

```
 -- ADD avg_one_day_score values
ALTER TABLE rcrs_rms_candidate_segment_flat_geom DROP COLUMN IF EXISTS avg_one_day_score;
ALTER TABLE rcrs_rms_candidate_segment_flat_geom ADD COLUMN avg_one_day_score varchar(5);

UPDATE  rcrs_rms_candidate_segment_flat_geom
SET avg_one_day_score =
CASE
WHEN avg_one_day <= 1.5 THEN  '10'
WHEN avg_one_day > 1.5 and avg_one_day <= 3 THEN  '7.5'
WHEN avg_one_day > 3 and avg_one_day <= 5 THEN  '5'
WHEN avg_one_day > 5 and avg_one_day <= 7 THEN  '2.5'
WHEN avg_one_day > 7 THEN '1'
END;


 -- ADD avg_four_day_score values

ALTER TABLE rcrs_rms_candidate_segment_flat_geom DROP COLUMN IF EXISTS avg_four_day_score;
ALTER TABLE rcrs_rms_candidate_segment_flat_geom ADD COLUMN avg_four_day_score varchar(5);

UPDATE  rcrs_rms_candidate_segment_flat_geom
```

```
SET avg_four_day_score =
CASE
WHEN avg_four_day <= 1.5 THEN  '10'
WHEN avg_four_day > 1.5 and avg_four_day <= 3 THEN  '7.5'
WHEN avg_four_day > 3 and avg_four_day <= 5 THEN  '5'
WHEN avg_four_day > 5 and avg_four_day <= 7 THEN  '2.5'
WHEN avg_four_day > 7 THEN '1'
END;



-- select * from rcrs_rms_candidate_segment_flat_geom;
```

- Is the segment in a floodplain?

```
-- simplify floodplain geometry and remove excess .kml fields
DROP TABLE IF EXISTS state_floodplain_simp;

CREATE TABLE state_floodplain_simp AS
SELECT gid, st_simplify(the_geom,.0001) as the_geom
FROM
state_floodplain;

-- spatial index on floodplain geom
CREATE INDEX sidx_state_floodplain_simp_geom
  ON public.state_floodplain_simp
  USING gist
  (the_geom);


-- CHECK Floodplain RMS candidate intersctions
ALTER TABLE rcrs_rms_candidate_segment_flat_geom DROP COLUMN IF EXISTS  in_floodplain;
ALTER TABLE rcrs_rms_candidate_segment_flat_geom ADD COLUMN in_floodplain varchar(5);

UPDATE rcrs_rms_candidate_segment_flat_geom a
SET in_floodplain = '10'
WHERE EXISTS
(
SELECT a.rms_key
FROM
state_floodplain_simp b
--rcrs_rms_candidate_segment_flat_geom a
WHERE
-- constrain by bounding box
a.the_geom && b.the_geom and
-- and then intersects test
```

```
st_intersects(a.the_geom, b.the_geom)
);


-- ADD zero to null in_floodplain
UPDATE rcrs_rms_candidate_segment_flat_geom
SET in_floodplain = '0'
WHERE
in_floodplain IS NULL;

-- select * from rcrs_rms_candidate_segment_flat_geom;
```

- Get and Set the OPI & IRI values from RMSSEG.

```
-- ADD RMSSEG  opi, iri
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS opi_rating,
ADD COLUMN opi_rating varchar(10),
DROP COLUMN IF EXISTS iri_rating,
ADD COLUMN iri_rating varchar(10)
;

UPDATE rcrs_rms_candidate_segment_flat_geom b
SET
opi_rating = subquery.opi_rating,
iri_rating = subquery.iri_rating
FROM
(
select
--distinct on (b.rms_key)
--b.rms_key,
a.cty_code || a.st_rt_no || a.seg_no as rms_key,
a.opi_rating, a.iri_rating
from
rms_seg a,
rcrs_rms_candidate_segment_flat_geom b
where
st_dwithin(ST_LineInterpolatePoint(b.the_geom,.5),a.the_geom,.0001)
) as subquery
where
b.rms_key = subquery.rms_key;


-- UPDATE RMSSEG Scores based on value
```

```
--select distinct(opi_rating) from tmp_rcrs_rms_candidate_segment_flat_geom;
--EXCELLENT, GOOD, FAIR, POOR

ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS opi_rating_score,
ADD COLUMN opi_rating_score varchar(10)
;

UPDATE rcrs_rms_candidate_segment_flat_geom
SET opi_rating_score =
CASE
WHEN opi_rating = 'EXCELLENT' THEN  '2.5'
WHEN opi_rating = 'GOOD' THEN  '5.0'
WHEN opi_rating = 'FAIR' THEN  '7.5'
WHEN opi_rating = 'POOR' THEN  '10.0'
END;
```

- Get and Set the Scour Critical Bridge scores from BMS.

```
-- ADD Scour Critical Bridge
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS bms_scour_score,
ADD COLUMN bms_scour_score varchar(5);

UPDATE rcrs_rms_candidate_segment_flat_geom a
SET bms_scour_score = '10'
where exists
(
select
--distinct on (rcrs_id)
a.rms_key,
b.bridge_id,
b.scourcrit
from
bms b
where
a.rms_key = b.cty_code || b.st_rt_no || b.seg_no
and
b.scourcrit in ('0','1','2','3','4','5')
);

-- ADD zero to null bms_scour_score
UPDATE rcrs_rms_candidate_segment_flat_geom
```

```
SET bms_scour_score = '0'
WHERE
bms_scour_score IS NULL;
```

- Get and Set the RMSPIPE score.

```
-- ADD RMSPIPE Score
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS rms_pipe_score,
ADD COLUMN rms_pipe_score varchar(5);

UPDATE rcrs_rms_candidate_segment_flat_geom a
SET rms_pipe_score = '10'
where exists
(
select distinct on (a.rms_key)
a.rms_key, b.drain_stru
from
rms_pipe b
where
a.rms_key = b.cty_code || b.st_rt_no || b.seg_no
and
b.drain_stru in ('1','2','3')
);

-- ADD zero to null rms_pipe_score
UPDATE rcrs_rms_candidate_segment_flat_geom
SET rms_pipe_score = '0'
WHERE
rms_pipe_score IS NULL;
```

- Get and Set the RMS Traffic score.

```
-- ADD RMS Traffic  cur_aadt, adtt_cur
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
```

```
DROP COLUMN IF EXISTS cur_aadt,
ADD COLUMN cur_aadt varchar(10),
DROP COLUMN IF EXISTS adtt_cur,
ADD COLUMN adtt_cur varchar(10)
;

UPDATE rcrs_rms_candidate_segment_flat_geom b
SET
cur_aadt = subquery.cur_aadt,
adtt_cur = subquery.adtt_cur
FROM
(
select
--distinct on (b.rms_key)
b.rms_key,
--a.cty_code || a.st_rt_no || a.seg_no as rms_key,
a.cur_aadt, a.adtt_cur
from
rms_traf a,
rcrs_rms_candidate_segment_flat_geom b
where
st_dwithin(ST_LineInterpolatePoint(b.the_geom,.5),a.the_geom,.0001)

) as subquery
where
b.rms_key = subquery.rms_key;


-- ADD cur_aadt_score
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS cur_aadt_score,
ADD COLUMN cur_aadt_score varchar(10);

UPDATE  rcrs_rms_candidate_segment_flat_geom
SET cur_aadt_score =
CASE
WHEN cur_aadt::float < 5000 THEN  '1'
WHEN cur_aadt::float >= 5000 and cur_aadt::float < 10000   THEN  '2'
WHEN cur_aadt::float >= 10000 and cur_aadt::float < 15000   THEN  '3'
WHEN cur_aadt::float >= 15000 and cur_aadt::float < 20000   THEN  '4'
WHEN cur_aadt::float >= 20000 and cur_aadt::float < 25000   THEN  '5'
WHEN cur_aadt::float >= 25000 and cur_aadt::float < 30000   THEN  '6'
WHEN cur_aadt::float >= 30000 and cur_aadt::float < 35000   THEN  '7'
WHEN cur_aadt::float >= 35000 and cur_aadt::float < 40000   THEN  '8'
WHEN cur_aadt::float >= 40000 and cur_aadt::float < 45000   THEN  '9'
WHEN cur_aadt::float >= 45000   THEN  '10'
END;

-- ADD adtt_cur_score
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS adtt_cur_score,
ADD COLUMN adtt_cur_score varchar(10);
```

```
UPDATE  rcrs_rms_candidate_segment_flat_geom
SET adtt_cur_score =
CASE
WHEN adtt_cur::float < 500 THEN  '1'
WHEN adtt_cur::float >= 500 and adtt_cur::float < 1000   THEN  '2'
WHEN adtt_cur::float >= 1000 and adtt_cur::float < 2000   THEN  '3'
WHEN adtt_cur::float >= 2000 and adtt_cur::float < 3000   THEN  '4'
WHEN adtt_cur::float >= 3000 and adtt_cur::float < 4000   THEN  '5'
WHEN adtt_cur::float >= 4000 and adtt_cur::float < 5000   THEN  '6'
WHEN adtt_cur::float >= 5000 and adtt_cur::float < 6000   THEN  '7'
WHEN adtt_cur::float >= 6000 and adtt_cur::float < 7000   THEN  '8'
WHEN adtt_cur::float >= 7000 and adtt_cur::float < 8000   THEN  '9'
WHEN adtt_cur::float >= 8000  THEN  '10'
END;
```

- Get and Set the RMS Admin score.

```
--ADD RMS ADMIN

ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS fed_func_cls,
ADD COLUMN fed_func_cls varchar(10)
;

UPDATE rcrs_rms_candidate_segment_flat_geom b
SET
fed_func_cls = subquery.fhwa_func_
FROM
(
select
b.rms_key,
a.fhwa_func_
from
rms_admin a,
rcrs_rms_candidate_segment_flat_geom b
where
st_dwithin(ST_LineInterpolatePoint(b.the_geom,.5),a.the_geom,.0001)
) as subquery
where
b.rms_key = subquery.rms_key;

-- UPDATE RMSADMIN Func Clas score
ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS fed_func_cls_score,
ADD COLUMN fed_func_cls_score varchar(10)
;

UPDATE rcrs_rms_candidate_segment_flat_geom
SET fed_func_cls_score =
CASE
--Interstate
WHEN fed_func_cls = '1' THEN  '10'
--Principal Arterial
WHEN fed_func_cls = '2' THEN '7.5'
--Principal Arterial Other
WHEN fed_func_cls = '3' THEN  '5.5'
--Collection or Minor Arterial
WHEN fed_func_cls in('4','5','6') THEN  '2.5'
--Local
WHEN fed_func_cls = '7' THEN  '1'
-- not modeled
WHEN fed_func_cls = '0' THEN  '0'
-- not modeled
WHEN fed_func_cls = '9' THEN  '0'
END;
```

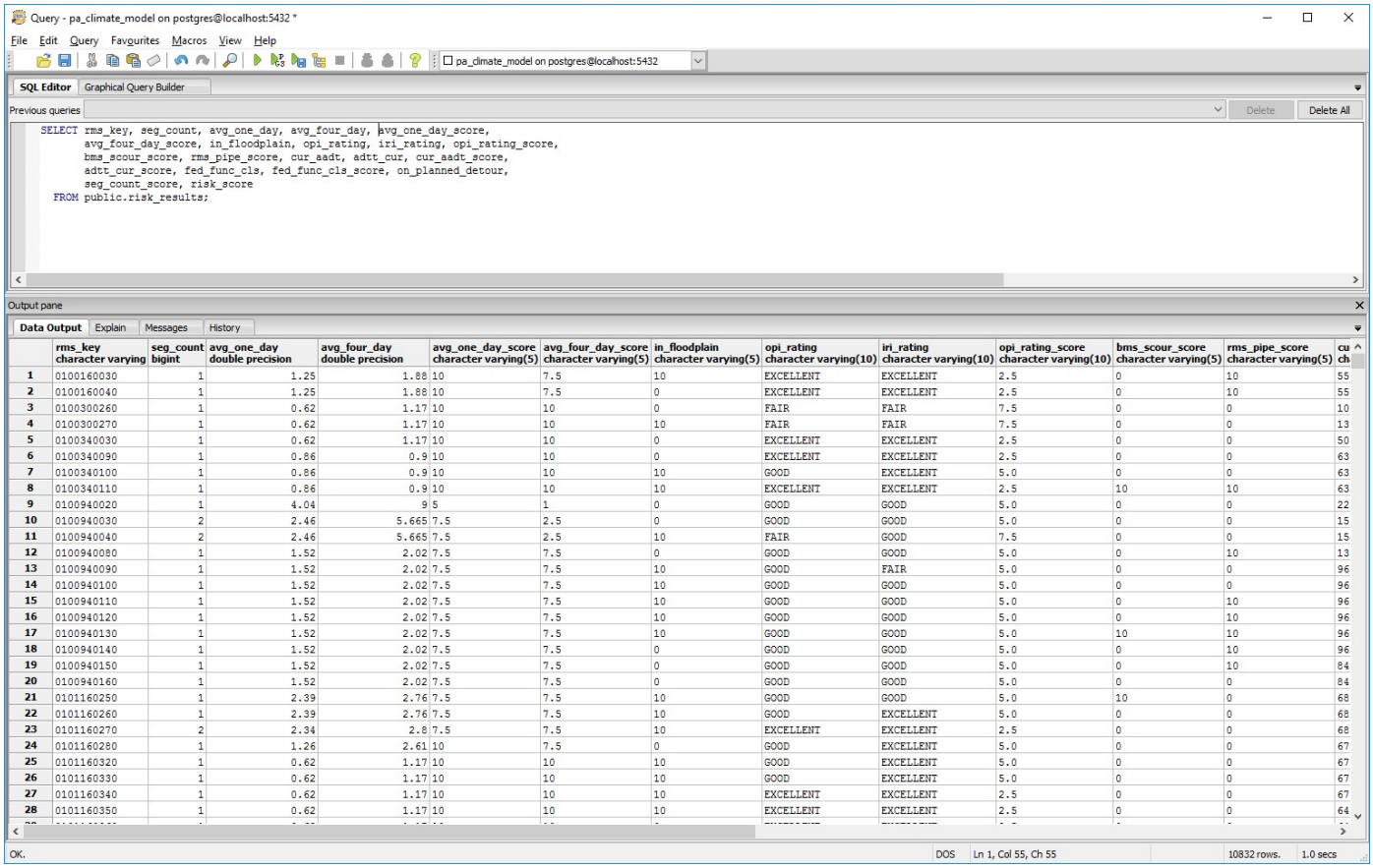
- Check if the segment is on a Planned Detour Route and score according to the model.

```
-- CHECK IF SEGMENTS ARE ON Planned Detour Route

/*
ALTER TABLE
tmp_rcrs_rms_candidate_segment_flat_geom
DROP COLUMN on_planned_detour;

*/


ALTER TABLE
rcrs_rms_candidate_segment_flat_geom
DROP COLUMN IF EXISTS on_planned_detour,
ADD COLUMN on_planned_detour varchar(5);

UPDATE rcrs_rms_candidate_segment_flat_geom a
SET on_planned_detour = '10'
where exists
(
select *
from
planned_detour b
where
st_dwithin(ST_LineInterpolatePoint(a.the_geom,.5),b.the_geom,.0001)
);

-- ADD zero to null on_planned_detour

UPDATE rcrs_rms_candidate_segment_flat_geom
SET on_planned_detour = '0'
WHERE
on_planned_detour IS NULL;
```

- Calculate the overall model score.

```
DROP TABLE IF EXISTS risk_score;

CREATE TABLE risk_score AS
SELECT
rms_key,
(3 * ((avg_four_day_score::float *.3) + (in_floodplain::float *.3 ) +
(seg_count_score::float * .4)   )) +
(3 * ((opi_rating_score ::float * .4) + (bms_scour_score::float * .4) +
(rms_pipe_score::float * .2))) +
(4 * ((cur_aadt_score::float * .4) + (fed_func_cls_score::float * .3) +
(on_planned_detour::float * .1) + (adtt_cur_score::float * .2)))
AS risk_score
FROM
rcrs_rms_candidate_segment_flat_geom;



--
DROP TABLE IF EXISTS risk_results;
CREATE TABLE tmp_risk_results AS
SELECT a.*, b.risk_score
FROM
rcrs_rms_candidate_segment_flat_geom a,
tmp_risk_score b
WHERE
a.rms_key = b.rms_key;

SELECT * FROM risk_results;
```

- The image below shows the final table of the data model, **risk_results**. This data can be mapped and further analyzed for insights.



Query - pa_climate_model on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

pa_climate_model on postgres@localhost:5432

SQL Editor | Graphical Query Builder

Previous queries | Delete | Delete All

```
SELECT rms_key, seg_count, avg_one_day, avg_four_day, avg_one_day_score,
       avg_four_day_score, in_floodplain, opi_rating, iri_rating, opi_rating_score,
       bms_scour_score, rms_pipe_score, cur_aadt, adtt_cur, cur_aadt_score,
       adtt_cur_score, fed_func_cls, fed_func_cls_score, on_planned_detour,
       seg_count_score, risk_score
  FROM public.risk_results;
```

Output pane

Data Output | Explain | Messages | History

	rms_key character varying	seg_count bigint	avg_one_day double precision	avg_four_day double precision	avg_one_day_score character varying(5)	avg_four_day_score character varying(5)	in_floodplain character varying(5)	opi_rating character varying(10)	iri_rating character varying(10)	opi_rating_score character varying(10)	bms_scour_score character varying(5)	rms_pipe_score character varying(5)	cu ch
1	0100160030	1	1.25	1.88	10	7.5	10	EXCELLENT	EXCELLENT	2.5	0	10	55
2	0100160040	1	1.25	1.88	10	7.5	0	EXCELLENT	EXCELLENT	2.5	0	10	55
3	0100300260	1	0.62	1.17	10	10	0	FAIR	FAIR	7.5	0	0	10
4	0100300270	1	0.62	1.17	10	10	10	FAIR	FAIR	7.5	0	0	13
5	0100340030	1	0.62	1.17	10	10	0	EXCELLENT	EXCELLENT	2.5	0	0	50
6	0100340090	1	0.86	0.9	10	10	0	EXCELLENT	EXCELLENT	2.5	0	0	63
7	0100340100	1	0.86	0.9	10	10	10	GOOD	EXCELLENT	5.0	0	0	63
8	0100340110	1	0.86	0.9	10	10	10	EXCELLENT	EXCELLENT	2.5	10	10	63
9	0100940020	1	4.04	9	5	1	0	GOOD	GOOD	5.0	0	0	22
10	0100940030	2	2.46	5.665	7.5	2.5	0	GOOD	GOOD	5.0	0	0	15
11	0100940040	2	2.46	5.665	7.5	2.5	10	FAIR	GOOD	7.5	0	0	15
12	0100940080	1	1.52	2.02	7.5	7.5	0	GOOD	GOOD	5.0	0	10	13
13	0100940090	1	1.52	2.02	7.5	7.5	10	GOOD	FAIR	5.0	0	0	96
14	0100940100	1	1.52	2.02	7.5	7.5	10	GOOD	GOOD	5.0	0	0	96
15	0100940110	1	1.52	2.02	7.5	7.5	10	GOOD	GOOD	5.0	0	10	96
16	0100940120	1	1.52	2.02	7.5	7.5	10	GOOD	GOOD	5.0	0	10	96
17	0100940130	1	1.52	2.02	7.5	7.5	10	GOOD	GOOD	5.0	10	10	96
18	0100940140	1	1.52	2.02	7.5	7.5	0	GOOD	GOOD	5.0	0	10	96
19	0100940150	1	1.52	2.02	7.5	7.5	0	GOOD	GOOD	5.0	0	10	84
20	0100940160	1	1.52	2.02	7.5	7.5	0	GOOD	GOOD	5.0	0	0	84
21	0101160250	1	2.39	2.76	7.5	7.5	10	GOOD	GOOD	5.0	10	0	68
22	0101160260	1	2.39	2.76	7.5	7.5	10	GOOD	EXCELLENT	5.0	0	0	68
23	0101160270	2	2.34	2.8	7.5	7.5	10	EXCELLENT	EXCELLENT	2.5	0	0	68
24	0101160280	1	1.26	2.61	10	7.5	0	GOOD	EXCELLENT	5.0	0	0	67
25	0101160320	1	0.62	1.17	10	10	10	GOOD	EXCELLENT	5.0	0	0	67
26	0101160330	1	0.62	1.17	10	10	10	GOOD	EXCELLENT	5.0	0	0	67
27	0101160340	1	0.62	1.17	10	10	10	EXCELLENT	EXCELLENT	2.5	0	0	67
28	0101160350	1	0.62	1.17	10	10	10	EXCELLENT	EXCELLENT	2.5	0	0	64

OK. | DOS | Ln 1, Col 55, Ch 55 | 10832 rows. | 1.0 secs

- The web maps contain all the results from the data model.